Sequential decomposition of operations and compilers optimization

Mumtaz Ahmad, Serge Burckel, Adam Cichon

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Sequential decomposition of operations and compilers optimization

Mumtaz AHMAD and Serge BURCKEL

N° 7076

October 2009

Thème SYM

ISSN 0249-6399 ISRN INRIA/RR--7076--FR+ENG

Sequential decomposition of operations and compilers optimization

Mumtaz AHMAD and Serge BURCKEL *

Thème SYM — Systèmes symboliques
Projet CASSIS

Rapport de recherche n° 7076 — October 2009 — 91 pages

Abstract: Code optimization is an important area of research that has remarkable contributions in addressing the challenges of information technology. It has introduced a new trend in hardware as well as in software. Efforts that have been made in this context led to introduce a new foundation, both for compilers and processors.
In this report we study different techniques used for sequential decomposition of mappings without using extra variables. We focus on finding and improving these techniques of computations. Especially, we are interested in developing methods and efficient heuristic algorithms to find the decompositions and implementing these methods in particular cases. We want to implement these methods in a compiler with an aim of optimizing code in machine language. It is always possible to calculate an operation related to K registers by a sequence of assignments using only these K registers. We verified the results and introduced new methods. We described In Situ computation of linear mapping by a sequence of linear assignments over the set of integers $\mathbb{Z}$ and investigated bound for the algorithm. We introduced a method for the case of boolean bijective mappings via algebraic operations over polynomials in $GF(2)$. We implemented these methods using Maple.

Key-words: Linear algebra, number theory, boolean mappings, polynomials, sequential computation, compiler / processor / memory optimization, in-place algorithms, algorithm performance, multi-objective search.

* LORIA & Universié Henri Poincaré. Nancy 1, (Mumtaz.Ahmad@inria.fr).

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,
615, rue du Jardin Botanique, BP 101, 54602 Villers-Lès-Nancy (France)
Téléphone : +33 3 83 59 30 00 — Télécopie : +33 3 83 27 83 19

Décomposition séquential des opérations pour l'optimisation des compilateurs

Résumé : L'optimisation de code est un domaine de recherche en plein essor et dont les contributions pour faire face aux défis inhérents aux technologies de l'information sont considérables. Elle a introduit une nouvelle tendance dans le matériel ainsi que dans le logiciel. Les efforts qui ont été réalisés dans ce contexte ont conduit à introduire de nouveaux fondements, à la fois pour les compilateurs et les processeurs.
Dans ce rapport, nous étudions différentes techniques de décomposition séquentielle *in situ* d'applications. Nous nous concentrons sur la recherche et l'amélioration de techniques de calcul et sur le développement d'heuristiques efficaces pour trouver ces décompositions, et mettons en œuvre ces méthodes de calcul dans un compilateur afin d'optimiser du code en langage machine. Il est toujours possible de calculer une opération reliée à K registres par une série d'affectations à l'aide de ces seuls K registres. Nous avons vérifié les résultats et introduit de nouvelles méthodes. Nous avons décrit le calcul *in situ* de la cartographie linéaire par une séquence d'affectations linéaires sur l'ensemble des entiers $\mathbb{Z}$ et étudié la complexité de l'algorithme. Nous avons introduit une méthode pour le cas des applications via des opérations booléennes bijectives algébriques sur les polynômes en $GF(2)$. Nous avons mis en œuvre ces méthodes en utilisant Maple.

Mots-clés : Algèbre linéaire, théorie des nombres, fonctions booléennes, polynômes, calcul séquentiel, optimisation de la mémoire / compilateur / processeur, algorithmes *in situ*, performance algorithmique, recherche multi-objectifs.

Contents

List of Figures

List of Tables

Chapter 1

Background

This chapter consists of three main sections. In the first section, we describe the basic concepts of compiler and processor and the necessary background which is required to understand the terminologies. The second section explains the motivations and objectives. The last section describes a number of applications of compilers and processors.

1.1 Introduction

In this section, we describe some basic concepts about compilers, processors, registers optimization, code optimization, techniques for compiler optimization, examples of compilers, types of optimizations and classification of microprocessors.

1.1.1 Compilers

A computer programmer typically writes softwares using high-level programming languages *e.g.* C++, Java etc. People can understand these languages but not directly the processors. Assembly language is the language that the processor can understand (the machine language). Two different pieces of code in assembly language could be equivalent in what they do, but perform this work using a different sequence of steps.
For example, when adding three numbers 1, 2 and 3 together, there are different ways, the computer could execute this. One way would be to add 1 and 2 together and then add 3 to that result $(1 + 2) + 3$. Another way to add the three numbers would be to add 2 and 3 together, and then add 1 to that result $(2 + 3) + 1$. A compiler has many choices in which specific implementation of assembly language it will choose in making the translation from the high-level programming language [43]. A high level language must be converted to a language that a processor can understand, this task is performed by the compiler. The compilers were introduced in the early $1950's$.

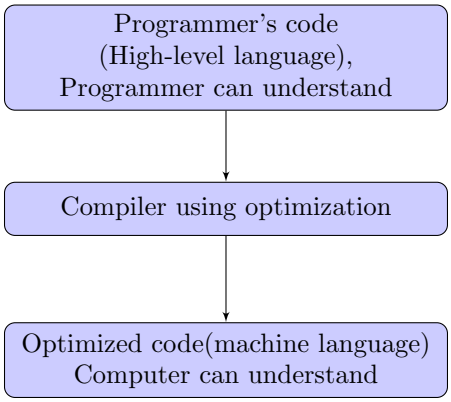

Figure 1.1: Complier interaction

A compiler (more generally, translator) is a software application that converts the code written by the programmer into machine language [18]. For example,

- gcc : converts C/C++ programs to assembly/machine code
- f2c : converts Fortran programs to C programs
- latex2html: converts Latex documents to HTML documents
- Java : converts Java programs to JVM byte code
- ps2pdf: converts Post Script files to PDF files

1.1.2 Compiler Optimization

The main purpose of compiler optimization is to reduce the time taken by the program to be executed. The process focuses also to minimize the usage of memory storage and power consumption. Compiler optimization enables the program to be more efficient and ultimately the whole process helps to increase the speed for compilation. Usually three types of optimizations are considered:

- Global optimization seeks to reorder the sequencing of a program in order to eliminate redundant computations (moving invariant operations outside loop bodies, colapsing loops, etc.).
- Register optimization adjusts the allocation of machine registers to variables and intermediate quantities in such a way as to minimize the number of a register has to be stored and later reloaded.

- Local (time) optimization seeks to adapt the code to exploit particular features of the machine architecture and to remove local mishandling such as loading a register with a value that it already contains.

There exist some code optimization problems that are considered to be NP-complete, or even undecidable [26, 20].

1.1.3 Compiler Optimization Techniques

There exist many compiler optimization techniques [48] but the work that has been done for the classification of these techniques can be improved. These methods can be divided into two main categories *i.e.* static and dynamic. These two approaches developed in parallel with a small interaction and optimization like parallelization has been obtained. Recently, the program parallelization of existing code came to forefront due to the fact that the major industrials focus to multi-core based architecture [47].

Static analysis of references to scalable data structures *e.g.* arrays can be very helpful, for the purpose of thread level parallelization. It helps in proving sufficient conditions to observe the independence about the behaviour of array indexes, solving linear integer equations with constraints [30], special cases of array data flow (privatization) and outright pattern matching.
Pointer analysis and shape analysis are also the important techniques but they are not very successful techniques [47].
Compiler optimization techniques operate on three levels: machine dependent, architecture dependent and independent. Instruction-level sensitivities of a compiler is described in the machine dependent. Architecture dependent includes parts of a program that relate to the general hardware implementation, but not to a specific machine. Architecture independent focus on that aspects of program formulation that do not depend on a particular computer system or even on a type of implementation (e.g. pipeline processing). Optimizations originating in the academic and scientific community tend to be global, while, until recently, manufacturers have concentrated on local and machine-dependent techniques [53, 42].
Some Programming techniques take advantage of the optimizing compilers and the system architecture, *e.g.* BLAS, a library of Basic Linear Algebra Subroutines. The subroutines included in this library are able to provide significant enhancement in the performance of a program that is numerically intensive.
ESSL, Engineering Scientific Subroutine Library. In fact, the ESSL library is an extension of BLAS library and includes high-performance mathematical routines for chemistry, engineering, and physics.
PESSL (a Parallel ESSL) is another library that exists for SMP machines. Compiler options and the use of preprocessors like KAP and VAST, available from third-party vendors [13].
Several run-time techniques have been introduced by computer architects on the basis of hardware. This enabled the processor that it can run any ready instruction from an instruction window [32, 35, 56]. Costly pipeline stalls has been prevented by bad execution and

detection of branches. Now the instruction can be stored in their dynamic execution order by using trace cache [21, 45]. Many run-time optimization has been introduced on the basis of hardware by latest microprocessors.

1.1.4 Processors

The processor, also known as the microprocessor (designed for microcomputers and micro controllers) or CPU (Central Processing Unit), is a complete computation engine that is integrated in a single chip. In fact, it is an integrated circuit, containing the arithmetic, logic, and control circuitry, and is used to interpret and execute instructions from a computer program.
This integrated circuit, in combination with other integrated circuits that provide memory to store and execute the program, form a chip.
Microprocessor registers used to hold temporary results, when the computation is being performed. Since these registers and microprocessors are made by the same technology, therefore there is no speed disparity between them. Moreover, these registers acts as processor memory.
To improve the performance of microprocessor a small memory has been introduced between microprocessor and main memory. This small memory is called cache memory, it is expensive but fast and is first time introduced in IBM 360/85 computer. The first microprocessor

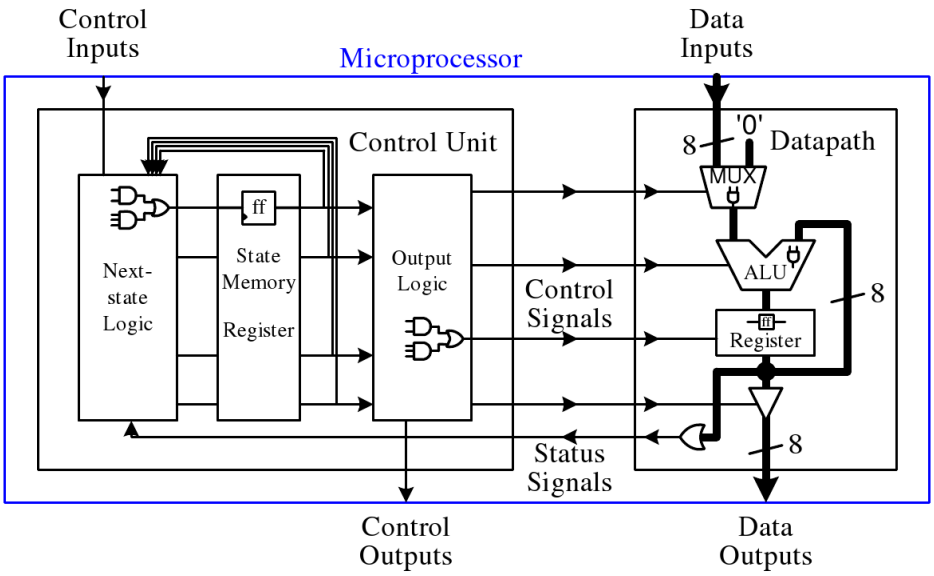

Figure 1.2: Microprocessor

was the Intel 4004, introduced in 1971. It was able to perform only subtraction and addition up to 4 bits at a time but everything was first time on a single chip.
The microprocessors can be classified on the basis of:

- the semiconductor technology of their design *e.g.* TTL, CMOS or ECL.
- the width of the data format (4, 8, 16, 32, 64-bit) they process
- their instruction set *e.g.* CISC or RISC

Due to the low power consumption, CMOS (complementary-metal-oxide semiconductor) technology is preferred to used in portable computers and in other devises that used batteries while TTL (transistor-transistor logic) is commonly used.

When high performance is needed, the ECL(emitter-coupled logic) is used. For example, Older high-end mainframe computers, like the Enterprise System/9000 members of IBM's ESA/390 computer family, used ECL but current IBM mainframes use CMOS [6].
Four-bit devices are good for simple control applications and they are not so costly.
CISC, (complex-instruction-set computer) processors, which have 70 to several hundred instructions, are easier to program than RISC, (reduced-instruction-set computer) processors, but are slower and more expensive [2, 28, 52, 55].

1.1.5 Processor optimization

How efficiently and effectively the processor executes instructions (provided in the form of a program designed by using some high level language) is determined by its internal design, also called its architecture. A processor can be considered as a combination of small blocks that organize to make a system. To help optimize the design space, a model is required that can predict the performance of a processor as a function of the delays of the underlying blocks. With such a model, one can evaluate how a change in the delay of a given module will affect the system's performance and can use this information to optimize a design. Because the design space is complicated, therefore, it may be difficult to know how changing the delay of a module will affect the overall performance of a processor [2].
Processor architect continue their efforts to improve the performance of processor every year. Some of the major techniques used by processor architects are the use of Wider data buses and registers, Floating point Units, Pipe lining and super scale architecture.
As processor speed continues to increase faster than memory speed, optimization to use the memory hierarchy efficiently become ever more important.
Blocking [24] or tiling [60] is a well-known technique that improves the data locality of numerical algorithms [1, 29, 31, 38, 40]. The improvement obtained from tiling can be far greater. Tiling can be used for different levels of memory hierarchy such as physical memory, caches and registers; multi-level tiling can be used to achieve locality in multiple levels of the memory hierarchy simultaneously [59]. The number of registers available on a processor and the operations that can be performed using those registers has a significant impact on the quality of code generated by optimizing compilers.

1.2 Applications

In recent years, the incorporation of computers into a large number of devices *e.g.* laptop, palmtop, telephone etc., has been increased significantly. In designing such devices the important factors that must be considered are space, weight, power consumption. But this will lead to the limited amount of memory availability.

A similar requirement is to use more and more sophisticated software in such devices, such as encryption software in telephones, or speech or image processing software in laptops and palm-tops.
It is not always possible to run an application on such devices that require memory more than the available. This makes it desirable to try to reduce the size of applications where possible [49].
A fast computer program (as a result of compiler optimization) is not only useful for computer scientist and computer architecture, it affects the general public as well. Compiler optimization helps to increase the efficiency and capabilities of not only the sophisticated software but also increase the demand of newly introduced computer based devices. Ultimately, compiler optimization directly affect the computer based technology used in particular as well as in common life.
For example, an improvement in computer programs for the medical community can affect all communities. An improved resolution of images obtained through scanning process has a direct affect on doctors and patients. An optimization will lead the life of the public to an ease.
A number of soft wares are being used only for the purpose of research to observe the output of different results. A maple program can take more than a week to see the output of a program, but if the same program yield the result within one hour or less, imagine, how fast will be the conclusions based on these results.

Microprocessors are the result of the semiconductor industrys ability to place an ever-greater number of transistors in a single integrated circuit. Optimized processors at the heart of mobile products enable communications, computing, and multimedia functions to be efficiently executed. Low power, high performance, and small form factor are key attributes. Open standards enable OS flexibility.
Microprocessors also play supporting roles within larger computers as smart controllers for graphics displays, storage devices, and high-speed printers.
However, the vast majority of microprocessors are used to control everything from consumer appliances to smart weapons. The microprocessor has made possible the inexpensive hand-held electronic calculator, the digital wristwatch, and the electronic game.
Microprocessors are used to control consumer electronic devices, such as the programmable microwave oven and DVD player; to regulate gasoline consumption and anti lock brakes in auto mobiles; to monitor alarm systems; and to operate automatic tracking and targeting systems in aircraft, tanks, and missiles and to control radar arrays that track and identify aircraft, among other defence applications. Intel®Core™ i7 processor is the most ever advanced desktop processor, introduced by Intel corporation recently in 2008. The Core

Figure 1.3: Processor Application

i7 processor is the first member of a new family of Nehalem processor designs and is the most sophisticated ever built, with new technologies that boost performance on demand and maximize data throughput. The Core i7 processor speeds video editing, immersion games and other popular Internet and computer activities by up to 40 percent without increasing power consumption.

Figure 1.4: Intel®Core™ i7 processor

1.3 Motivations and objectives

Since half a century, code optimization is considered to be an important area of research. It has introduced a new trend in hardware as well as in softwares. Efforts that have been made in this context led to introduce a new foundation, both for compilers and processors. New ideas have been introduced *e.g.* inter procedural whole program analysis, coloring-based register allocation, static single assignment form, array dependence analysis, pointer alias analysis, loop transformations, adaptive profile-directed optimizations, and dynamic compilation.
Multi core processors have been introduced almost in all new computers. This multi core trend in the computer industry is forcing a new paradigm shift in compilers to face new challenges [50, 51]. The code of the program that is to be compiled has high effect on optimization. A large number of optimization has been introduced by modern compilers. These compilers have different effect on quality and size of code, time taken and energy consumption etc. [27].
In [7, 8, 9, 10, 11], it is shown that it is always possible to calculate an operation related to K registers by a sequence of assignments using only these K registers. Moreover, if this operation is linear or bijective, the number of assignments is at most $2k$. In the general case, this number of assignments is at most $4k$.
For example, the bijection E on 3 bits defined by

$$E(A,\, B,\, C) = (1 + A + C + AB,\, A,\, B + AB + AC)$$

can be calculated by a sequence of 4 assignments as given below:

$$\begin{aligned} A &:= A + B + C \\ B &:= A + B + C \\ C &:= A + C + AB \\ A &:= 1 + A + C + BC \end{aligned}$$

Such kinds of calculations generalize the traditional principle of the exchange of two bits A, B by the sequence:

$$\begin{aligned} A &:= A + B \\ B &:= A - B \\ A &:= A - B \end{aligned}$$

We are intended to study these methods and to improve these techniques of calculations, and especially to develop methods and efficient heuristic algorithms to find these decompositions and implement these methods in compilers.

We are interested in improving the bounds of these calculations, to obtain new methods of calculation for particular cases and implementing these methods in a compiler with an aim of optimizing code in machine language.

It is a fact that current computer architectures reach their theoretical limits of performance. However, it is still possible to gain performance of calculations by performing calculations in a new way.
This implementation can be provided in hardware through the design of new processors, in the software, by optimizing compilers upstream (pre-level language compilers) and downstream (post-machine language compilers).
The current architectures use one or more processors equipped each one with a relatively low number of registers.
However, these registers are constantly requested in the operations. It is often advisable to minimize the use of registers.

Chapter 2

IN SITU Computations

The conversion of input into output under well defined sequence of basic computational steps leads to the theory of sequential computations. A sequential program implements a mathematical function that maps a set of inputs to a set of outputs. These mathematical functions are well defined. The notion of computable functions has been introduced earlier by Church, Kleene and Turing. These functions are frequently used in untyped lambda calculus, recursive functions, and Turing machines [17, 33, 57].
Although, these basic models help in designing and reasoning for programming languages, domain theory and denotational semantics introduced by Scott and Strachey, and provide a global mathematical setting for sequential computation, building on top of the foundational theories [54]. This interconnects different programming languages and makes connection with the mathematical world of algebra, topology, and logic. It inspires the programming languages, type disciplines, and reasoning methods.

2.1 In-place (In-situ) Algorithm:

An in-place algorithm converts data structure using a minimal constant extra storage space. When such algorithms run, the input is overwritten by the output. For example, heap sort is an in-place sorting algorithm.
An operation is said in-place operation if it does not alter the normal state of the system like a file backup can be stored over a running system without altering the speed of the system, while an in-place operation depends on the sophistication of the system.
In order to improve cache performance, an algorithm or application should increase data reuse, decrease cache conflicts, and decrease cache pollution. A large amount of cache pollution will increase the bandwidth requirement of the application, even though the application is not using more data [41].

In the next example, two linear assignments are being reused and modified repeatedly to compute a linear mapping E.

Example 1. *The mapping* $E : (x, y) \longrightarrow (377x + 610y, 610x + 987y)$
can be computed by the following eight assignments:

$$\begin{aligned} x &:= -x - 2y \\ y &:= -3x - y \\ x &:= -x - 3y \\ y &:= -3x - y \\ x &:= -x - 3y \\ y &:= -3x - y \\ x &:= x + 3y \\ y &:= 2x - y \end{aligned}$$

Observe that, the sequence of eight assignments is actually the repetition of two operations. The main advantage of such computation is the minimum usage of memory available.

2.2 In Situ Decomposition of linear mappings:

A linear mapping $E : S^n \longrightarrow S^n$ can be decomposed into a sequence of mappings $f : S^n \longrightarrow S$. The mapping E can be written as $E : X := AX$, where $A = [a_{i,j}]$ is a square matrix of size n, with $a_{i,j} \in \mathbb{R}$ and X is a vector with n variables. Each mapping $f : S^n \longrightarrow S$ could be considered as a linear assignment. These assignments can be computed by an in situ program that require no extra variables other than input variables. This in situ program can be simply written as follows:
for $i = 1$ *to* k *do*
$\quad x_{p_i} := a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n$
enddo
where $x_1, x_2, \cdots, x_n$ are the input variables and the output variables. The length of program is determined by the number β of assignments.

Example 2. *.Consider a mapping* E *that can be written as* $E : X := AX$*, such that*

$$X := \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} \text{ and } A := \begin{pmatrix} 2 & 3 & 5 & 8 \\ 3 & 5 & 8 & 13 \\ 5 & 8 & 13 & 21 \\ 8 & 13 & 21 & 34 \end{pmatrix}$$

The given mapping E can be decomposed into the following sequence of 6 *linear assignments:*

$$
\begin{aligned}
x_1 &:= x_1 + x_2 + 2x_3 + 3x_4 \\
x_2 &:= x_1 + x_2 + x_3 + 2x_4 \\
x_3 &:= 2x_1 + 3x_2 \\
x_4 &:= x_3 + x_1 + 2x_2 \\
x_1 &:= x_3 - x_1 - 2x_2 \\
x_2 &:= x_3 - x_1
\end{aligned}
$$

Observe that, the given mapping has been decomposed into a sequence of linear mappings in a way that, no extra variable other than the input variables is used.

2.3 State of the Art:-

In recent years, many computer scientists investigated the area of optimization including compiler optimization, processor optimization, mization, algorithm optimization etc. The optimization of specific linear algebra problems has been discussed on a large scale because such kinds of optimization effect on processor performance. Whaley and Dongarra discuss optimizing the widely used Basic Linear Algebra Subroutines (BLAS) in [58].
Chatterjee et al. discuss layout optimizations for a suite of dense matrix kernels in [16]. Park et al. discuss dynamic data remapping to improve cache performance for the DFT in [44]. Frigo, et al. in [22], which discusses the cache performance of cache oblivious algorithms for the matrix transpose, FFT, and sorting. Optimizing blocked algorithms has been extensively studied [36].

Most of the optimization algorithms depend on heuristics and approximations because several problems of optimizations are NP-complete. It may happen that a particular algorithm fails to produce better code or even worse. Code optimization transform a piece of code to make it more efficient without changing its output or side-effects [12].

An important optimization, affecting the performance of compiler code, is register allocation. Register allocation has been studied extensively in compilation and is a NP-complete problem. In 1981, Chaitin et al [14, 15] modelled the problem of assigning temporary variables to k machine registers as the problem of colouring, with k colours the interference graph associated to the variables.

In general, register access is faster than memory access. Hence it is preferable to use register than memory whenever it is possible. When it is not possible to use register then some variables must be transferred to memory. This load/store operation has its cost. To avoid this cost some classical approaches have been introduced like in graph colouring algorithms [4].

An iterated register coalescing algorithm, proposed by Appel and George [23] is a modified version of previous developments by Chaitin et al. [14, 15], and Briggs et al. [5]. In these heuristics, spilling, coalescing (removing register-to-register moves), and colouring (assigning variables to registers) are done in the same framework. These techniques are very useful in compiler optimization but still have to be revised to get better results.

Burckel et al. [8] have introduced methods to design programs/electronic circuits, for performing any operation on k registers of any sizes in a processor, in such a way that one uses no other working memory (such as other registers or external memories). In this way, any operation is performed with at most $4k-3$ assignments of these registers, or $2k-1$ when the operation is linear or bijective. These methods are directly connected with processor and compiler optimization.

In [7], it is proved that any mapping on $\{0,1\}^n$ can be computed by an in situ calculus without using extra variable. It is proved that three types of assignments are sufficient to perform these computations [9]. In [10], it is proved that any mapping E on $\{0,1\}^n$ has a sequential computation in at most n^2 steps. In [11], it is proved that any linear mapping of n dimension can be computed with a sequence of $2n-1$ linear assignments. It is also proved that every mapping E on S^n can be computed by an in situ program of length $5n-4$. The bound has been improved up to $4n-3$ when S is a power of 2. The maximal length of the program is $2n-1$ for bijective mappings [8].

2.4 Sequential Computation over $\mathbb{R}$:

In this section, we describe the proof that linear mappings on any field $\mathbb{R}$ admit sequential computation. Next, we explain, how the algorithm computes linear mappings sequentially over the field $\mathbb{R}$. To illustrate completely the ideas, we quoted different examples. We also quote some examples to explain, how the inverse mappings can be computed. We will also introduce an alternative approach to compute a linear mapping of dimension 2. We can also compute inverse mappings using this approach. We explain this approach by giving different examples.

Proposition 1. *Let E be a linear mapping on K^n, where K is a field and n is a positive integer. There exists a sequence $f_1, f_2, \cdots, f_{n-1}, f_n, g_{n-1}, \cdots, g_2, g_1$ of linear mappings from K^n to K such that the program :*

$$\begin{aligned}
x_1 &:= f_1(x_1, x_2, \cdots, x_n)\\
x_2 &:= f_2(x_1, x_2, \cdots, x_n)\\
\vdots &:= \quad\vdots \quad\quad \vdots\\
x_{n-1} &:= f_{n-1}(x_1, x_2, \cdots, x_n)\\
x_n &:= f_n(x_1, x_2, \cdots, x_n)\\
x_{n-1} &:= g_{n-1}(x_1, x_2, \cdots, x_n)\\
\vdots &:= \quad\vdots \quad\quad \vdots\\
x_2 &:= g_2(x_1, x_2, \cdots, x_n)
\end{aligned}$$

$$x_1 := g_1(x_1, x_2, \cdots, x_n)$$

performs the operation $X := E(X)$ *for any* $X = (x_1, x_2, \cdots, x_n)$ *in* K^n .
In addition:
For any i, g_i *will be* x_i *or* $x_i + x_j$ *for some* $j > i$.
E *will be bijective* $\Longleftrightarrow$ $f_i[i] \neq 0$ *for any* i.
For any i, *if* $f_i[i] = 0$ *then* $g_i = x_i$ *and* $f_j[i] = 0$ *for any* $j \geq i$.

Proof. The mapping $E := E(X)$ can be considered as $X := AX$, where A is a matrix of coefficients *i.e.* $a_{ji} \in K$, for $i, j = 1, 2, \cdots, n$. Each row of the matrix AX represents the components of mapping E at current values of the variables $(x_1, x_2, \cdots, x_n)$ and is considered to be a linear mapping . For example, the jth row $a_{j1} + a_{j2} + a_{j3} + \ldots + a_{jn}$, of matrix AX will be the E_j component of mapping E. These linear mappings can be denoted by F_i for $i = 1, 2 \cdots, n$. To manage the second sequence, the integers r_i can be introduced. The first sequence f_i can be computed by an iterative procedure as given below:
For i from 1 to n, the following steps will be performed keeping $r_i = i$ at the beginning.

> Case 1: If $a_{ii} = \alpha \neq 0$, then $f_i := F_i$, and x_i will be modified to $x_i := \alpha x_i + \Delta$, where Δ does not depend on x_i. For any $a_{ji} \neq 0, j > i$, we will use the reference $\alpha^{-1}(x_i - \Delta)$ to compute next mappings.
>
> Case 2: If $a_{ji} = \alpha = 0$, $\forall j \geq i$, then $f_i := F_i$, and $x_i := \Delta$, where Δ does not depend on x_i. Since $a_{ji} = 0$, $\forall j \geq i$, therefore no reference of x_i will be required for the next steps.
>
> Case 3: If $a_{ii} = \alpha = 0$, and $a_{ji} \neq 0$ for some $j > i$, then we can select $a_{ji} = \beta$ such that $\beta \neq 0$. We will perform the operation $F_i := F_i - F_j$ *i.e.* subtract jth row from ith row so that $\acute{\alpha} = -\beta \neq 0$, where $\acute{\alpha}$ denotes the new value of α. In this way, we will be again in case 1 and by the hypothesis the value $E_i - E_j$ will be assigned to x_i.
> The operation $F_i := F_i - F_j$ will introduce an assignment to the second sequence that can be obtained by adding E_j to x_i with $r_i := j$.

Computation of the second sequence:
The integers r_i, that has been introduced, will be used to build the second sequence of assignments.
Therefore from $i = n$ *to* 1, the following steps will be performed iteratively.

> Step 1: If $r_i = i$, then x_i has been assigned in the first sequence and nothing to do more.
>
> Step 2: If $r_i = j$ such that $j > i$. Then because it results by the operation $F_i := F_i - F_j$, therefore $g_i := x_i + x_j$, but for $i = n$, this situation will not exist.

The mapping E will not be injective in the Case-2 of the computation of first sequence but if $a_{ii} \neq 0$ for any i, then we can compute E^{-1} as well.
Write the assignments, obtained, from last to first.

Compute $x_i := \alpha^{-1}(x_i - \Delta)$ from $x_i := \alpha x_i + \Delta$, $(\alpha \neq 0)$. Use these references in the next assignments so that E^{-1} will be computed. □

2.4.1 Explanation:

In this section, we will explain, the above proposition. We explain different cases by giving different examples for each case separately.

Example 3. *Consider a linear mapping E of the form*

$$E\,(x_1,\, x_2,\, x_3) \longrightarrow (3x_1 + 7x_2 + 5x_3,\, 8x_1 + 4x_2 + 9x_3,\, 2x_1 + x_2 + 6x_3)$$

We know that, for a vector $X := (x_1, x_2, x_3)$, the mapping E can be written as $X := AX$, where A denotes the matrix of coefficients. Therefore, for the given mapping E, matrix A will be of the form as given below.

$$A = \begin{pmatrix} 3 & 7 & 5 \\ 8 & 4 & 9 \\ 2 & 1 & 6 \end{pmatrix}$$

Since $\alpha = 3 \neq 0$. Therefore, the first assignment is:

$$x_1 := 3x_1 + 7x_2 + 5x_3 \tag{2.1}$$

The initial value of x_1 will be

$$\frac{1}{3}(x_1 - 7x_2 - 5x_3)$$

Now, for $\beta = 8$, we perform the operation

$$x_2 := x_2 - \beta x_1 + \beta\left\{\alpha^{-1}\,(x_1 - \Delta)\right\}$$

on second row and for $\beta = 2$, we perform the operation

$$x_3 := x_3 - \beta x_1 + \beta\left\{\alpha^{-1}\,(x_1 - \Delta)\right\}$$

on third row of matrix A.

So, $x_2 := 8x_1 + 4x_2 + 9x_3 - 8x_1 + 8\left\{\dfrac{1}{3}(x_1 - 7x_2 - 5x_3)\right\}$.

$$x_2 := \frac{8}{3}x_1 - \frac{44}{3}x_2 - \frac{13}{3}x_3 \tag{2.2}$$

and $x_3 := 2x_1 + x_2 + 6x_3 - 2x_1 + \dfrac{2}{3}\,(x_1 - 7x_2 - 5x_3)$

$$x_3 := \frac{2}{3}x_1 - \frac{11}{3}x_2 + \frac{8}{3}x_3$$

After performing this operation the matrix A will take the form

$$A = \begin{pmatrix} 3 & 7 & 5 \\ \frac{8}{3} & -\frac{44}{3} & -\frac{13}{3} \\ \frac{2}{3} & -\frac{11}{3} & \frac{8}{3} \end{pmatrix}$$

Now, the initial value of second assignment, x_2 will be,

$$\frac{8}{44}x_1 - \frac{3}{44}x_2 - \frac{13}{44}x_3$$

The computation of third assignment:
For, $\alpha = -\frac{44}{3}$ and $\beta = -\frac{11}{3}$, We perform the following operation:

$$x_3 := x_3 - \beta x_2 + \beta \left\{ \alpha^{-1} \left(x_2 - \Delta \right) \right\}$$

$$x_3 := \frac{2}{3}x_1 - \frac{11}{3}x_2 + \frac{8}{3}x_3 + \frac{11}{3}x_2 - \frac{11}{3} \left\{ \frac{8}{44}x_1 - \frac{3}{44}x_2 - \frac{13}{44}x_3) \right\}$$

$$x_3 := \frac{1}{4}x_2 + \frac{15}{4}x_3 \tag{2.3}$$

Combining (2.1), (2.2) and (2.3), we get the required assignments, that are

$$\begin{aligned} x_1 &:= 3x_1 + 7x_2 + 5x_3 \\ x_2 &:= \frac{8}{3}x_1 - \frac{44}{3}x_2 - \frac{13}{3}x_3 \\ x_3 &:= \frac{1}{4}x_2 + \frac{15}{4}x_3 \end{aligned}$$

Computing the Inverse of Matrix A:

For a non-singular matrix A, we can also compute A^{-1}, by a sequence of assignments. We write, the sequence of assignments that can compute mapping E, from bottom to top, each assignment $x_i := \alpha x_i + \Delta$ as $x_i := \alpha^{-1} \{x_i - \Delta\}$. Applying this technique on the sequence of assignments

$$\begin{aligned} x_1 &:= 3x_1 + 7x_2 + 5x_3 \\ x_2 &:= \frac{8}{3}x_1 - \frac{44}{3}x_2 - \frac{13}{3}x_3 \\ x_3 &:= \frac{1}{4}x_2 + \frac{15}{4}x_3 \end{aligned}$$

we get the corresponding sequence of assignments that can compute A^{-1} and consequently, the mapping E^{-1}.

$$x_3 := \frac{4}{15}x_3 - \frac{1}{15}x_2$$
$$x_2 := \frac{8}{44}x_1 - \frac{3}{44}x_2 - \frac{13}{44}x_3$$
$$x_1 := \frac{1}{3}x_1 - \frac{7}{3}x_2 - \frac{5}{3}x_3$$

$$A^{-1} = \begin{pmatrix} \frac{-1}{11} & \frac{37}{165} & -\frac{43}{165} \\ \frac{2}{11} & -\frac{8}{165} & -\frac{13}{165} \\ 0 & -\frac{1}{15} & \frac{4}{15} \end{pmatrix}$$

Example 4. *Consider a linear mapping*

$$E\,(x_1,\, x_2,\, x_3) \longrightarrow (5x_2 + 7x_3,\, x_2 + 4x_3,\, 3x_1 + 2x_3)$$

For a vector $X := (x_1, x_2, x_3)$, the mapping E can be written as $X := AX$, where A is a matrix of coefficients. Therefore matrix A will be.

$$A = \begin{pmatrix} 0 & 5 & 7 \\ 0 & 1 & 4 \\ 3 & 0 & 2 \end{pmatrix}$$

Since $a_{11} = 0 = \alpha$. But $a_{13} = 3 = \beta$. Therefore we have to subtract third row of matrix A from first row. The operation will be $R_1 := R_1 - R_3$.
After the completion of this operation, the matrix A will take the form:

$$A = \begin{pmatrix} -3 & 5 & 5 \\ 0 & 1 & 4 \\ 3 & 0 & 2 \end{pmatrix}$$

Now the first assignment will be of the form:

$$x_1 := -3x_1 + 5x_2 + 5x_3 \tag{2.4}$$

The initial value of x_1 will be

$$\frac{1}{3}(-x_1 + 5x_2 + 5x_3)$$

Now, we will perform the operation

$$x_2 := x_2 - \beta x_1 + \beta\left\{\alpha^{-1}\,(x_1 - \Delta)\right\}$$

and

$$x_3 := x_3 - \beta x_1 + \beta \left\{ \alpha^{-1} \left(x_1 - \Delta \right) \right\}$$

The matrix A will take the form

$$A = \begin{pmatrix} -3 & 5 & 5 \\ 0 & 1 & 4 \\ -1 & 5 & 7 \end{pmatrix}$$

The second assignment will be

$$x_2 := x_2 + 4x_3 \tag{2.5}$$

Now, the initial value of second assignment, x_2 will be

$$x_2 - 4x_3$$

And for the computation of third assignment:
For, $\alpha = 1$ and $\beta = 5$, We perform the following operation:

$$x_3 := x_3 - \beta x_2 + \beta \left\{ \alpha^{-1} \left(x_2 - \Delta \right) \right\}$$

We will get the third assignment as

$$x_3 := -x_1 + 5x_2 - 13x_3 \tag{2.6}$$

The first assignment of second sequence that is obtained by the operation $R_1 := R_1 - R_3$ will be.

$$x_1 := x_1 + x_3$$

Hence the required assignments are

$$\begin{aligned} x_1 &:= -3x_1 + 5x_2 + 5x_3 \\ x_2 &:= x_2 + 4x_3 \\ x_3 &:= -x_1 + 5x_2 - 13x_3 \\ x_1 &:= x_1 + x_3 \end{aligned}$$

Computing Inverse of Matrix A:

We can compute A^{-1} by applying the same technique as described before i.e. by inverting the assignments and rewriting from bottom to top.

$$\begin{aligned} x_1 &:= x_1 - x_3 \\ x_3 &:= \frac{1}{13} \left(-x_1 + 5x_2 - x_3 \right) \\ x_2 &:= \frac{4}{13} x_1 - \frac{7}{13} x_2 \\ x_1 &:= \frac{1}{3} \left(-x_1 + 5x_2 + 5x_3 \right) \end{aligned}$$

$$A^{-1} = \begin{pmatrix} \frac{2}{39} & -\frac{10}{39} & \frac{1}{3} \\ \frac{4}{13} & -\frac{7}{13} & 0 \\ -\frac{1}{13} & \frac{5}{13} & 0 \end{pmatrix}$$

Example 5. *Consider a linear mapping*

$$E\,(x_1,\, x_2,\, x_3) \longrightarrow (3x_1 + 5x_2 + 7x_3,\, x_2 + 4x_3,\, 2x_3)$$

For a vector $X := (x_1, x_2, x_3)$, the mapping E can be written as $X := AX$, where A is a matrix of coefficients. Therefore matrix A will be.

$$A = \begin{pmatrix} 3 & 5 & 7 \\ 0 & 1 & 4 \\ 0 & 0 & 2 \end{pmatrix}$$

Since $a_{11} = 3 \neq 0 = \alpha$. Therefore the first assignment will be of the form:

$$x_1 := 3x_1 + 5x_2 + 7x_3 \tag{2.7}$$

The initial value of x_1 will be

$$\frac{1}{3}(x_1 - 5x_2 - 7x_3)$$

Since $\beta = 0$ both for second and third row, therefore, Matrix A will remain unchanged, and no reference of x_1 is used to compute the second assignment.
The second assignment will be

$$x_2 := x_2 + 4x_3 \tag{2.8}$$

In computation of third assignment, we also do not need to use any reference of first or second assignment. Therefore, for such kinds of mappings the sequence of assignments can be written directly from the mapping.

$$x_3 := 2x_3 \tag{2.9}$$

Hence the required assignments are

$$\begin{aligned} x_1 &:= 3x_1 + 5x_2 + 7x_3 \\ x_2 &:= x_2 + 4x_3 \\ x_3 &:= 2x_3 \end{aligned}$$

Computing Inverse of Matrix A:

The three assignments that can compute the inverse of matrix A can be written by applying the technique of inverting and rewriting the assignments from bottom to top:

$$x_3 := \frac{1}{2}x_3$$
$$x_2 := x_2 - 4x_3$$
$$x_1 = \frac{1}{3}(x_1 - 5x_2 - 7x_3)$$

$$A^{-1} = \begin{pmatrix} \frac{1}{3} & -\frac{5}{3} & \frac{13}{6} \\ 0 & 1 & -2 \\ 0 & 0 & \frac{1}{2} \end{pmatrix}$$

Example 6. *Consider a linear mapping*

$$E\,(x_1,\, x_2,\, x_3) \longrightarrow (5x_2 + 7x_3,\, x_2 + 4x_3,\, 2x_3)$$

For a vector $X := (x_1, x_2, x_3)$, the mapping E can be written as $X := AX$, where A is a matrix of coefficients. Therefore matrix A will be.

$$A = \begin{pmatrix} 0 & 5 & 7 \\ 0 & 1 & 4 \\ 0 & 0 & 2 \end{pmatrix}$$

Since $a_{11} = 0 = \alpha$ and $\beta = 0$ for all other cases, therefore we do not need to perform any operation, and the required assignments can be written directly from the given mapping or from the matrix A.

$$x_1 := 5x_2 + 7x_3$$
$$x_2 := x_2 + 4x_3$$
$$x_3 := 2x_3$$

Computing Inverse of Matrix A:

Since the matrix A is a singular, therefore A^{-1} does not exist. It is also obvious that the assignment

$$x_1 := 5x_2 + 7x_3$$

is not an invertible assignment.

2.4.2 An alternative approach:

Lemma 1. *Every linear mapping E:* $(x, y) \longrightarrow (mx + ny, px + qy)$*, where* $m, n, p, q \in \mathbb{Z}$*, can be computed by a sequence of at most* 3 *linear assignments with rational coefficients.*

Proof. Consider a sequence of three linear assignments:

$$x := ax + by$$

$$y := cx + dy$$

$$x := ex + fy$$

Where a, b, c, d, e, and f are rational In sequential computation, the linear assignments will take the form:

$$\begin{aligned} x &:= ax + by \\ y &:= cax + (cb + d)y \\ x &:= (ea + fca)x + (eb + fcb + fd)y \end{aligned}$$

The given Linear mapping $E(x, y) \longrightarrow (mx + ny, px + qy)$, can be viewed as a matrix of order 2×2 as below.

$$\begin{pmatrix} m & n \\ p & q \end{pmatrix} = A(say)$$

We can establish a system of four equations as given below:

$$\begin{aligned} m &= ea + fca \\ n &= eb + fcb + fd \\ p &= ca \\ q &= bc + d \end{aligned} \tag{5}$$

Moreover, $Determinant(A) = m * q - p * n = a * e * d$
$\Longrightarrow$ The product of three variables a, e and d should must be equal to the Determinant of matrix A. Therefore, a, e and d, will be the factors of Determinant of matrix A.
The general solution of system (5) of equations is:

$$a = -\frac{fp - m}{e},\ b = -\frac{fq - n}{e},\ c = -\frac{pe}{fp - m},\ d = -\frac{mq - np}{fp - m} \tag{2.10}$$

Bézout's identity states that, if two integers a and b are relatively prime then there exist u, $v \in \mathbb{Z}$ such that $au + bv = 1$.
Using Bézout's identity, we can construct a new matrix, such that we can compute the mapping.
Suppose that m and p are relatively prime *i.e.* $gcd(m, p) = 1$.
Using Bézout's identity, we can find u, $v \in \mathbb{Z}$ such that $um + vp = 1$

Multiply first row of the matrix A, by u and second by v, so that, u and v will be the common factor of entries of first and second row respectively, the new matrix will satisfy the required conditions.
We will return to the original matrix by performing the operations.
$y := (\frac{1}{v})y$ and $x := (\frac{1}{u})x$
If m and p are not Co prime, then we can make m and p Co prime by extracting their gcd. We will solve the mapping as, for Co prime case and finally we will assign the GCD to a. □

The above lemma can be explained by the following examples.

Example 7. *Consider the linear mapping*

$$E : (x,\, y) \longrightarrow (55x + 89y,\, 34x + 21y)$$

such that matrix A will represent the coefficients.

$$A = \begin{pmatrix} m & n \\ p & q \end{pmatrix} = \begin{pmatrix} 55 & 89 \\ 34 & 21 \end{pmatrix}$$

Since $GCD(55, 89) = 1$*, Therefore, we can apply Bézout's identity to find integers u and v such that* $55u + 89v = 1$
A number of solutions is possible for u and v. One of them is $u = 13$, $v = -21$
Multiply first row of matrix A by u and second by v, we will get

$$\acute{A} = \begin{pmatrix} um & un \\ vp & vq \end{pmatrix} = \begin{pmatrix} 715 & 1157 \\ -714 & -441 \end{pmatrix}$$

Using

$$a = -\frac{fp - m}{e},\; b = -\frac{fq - n}{e},\; c = -\frac{pe}{fp - m},\; d = -\frac{mq - np}{fp - m} \tag{2.11}$$

we will have $a = 1$, $b = 716$, $c = -714$, $d = 510783$, $e = 1$, $f = -1$
Hence the sequence of assignments is as given below:

$$\begin{aligned} x &:= x + 716y \\ y &:= \frac{1}{13}\,(-714x + 510783y) \\ x &:= -\frac{1}{21}\,(x - y) \end{aligned}$$

2.4.3 Computing the Inverse Mapping:

We can also compute the inverse mapping using this alternate approach. We will explain the idea by giving different examples as follows.

Example 8. *In fact, the sequence of assignments that compute the mapping E represented by a matrix*

$$\acute{A} = \begin{pmatrix} 715 & 1157 \\ -714 & -441 \end{pmatrix}$$

is given by

$$\begin{aligned} x &:= x + 716y \\ y &:= -714x + 510783y \\ x &:= x - y \end{aligned}$$

The sequence of assignments that can compute the $\acute{A}^{-1}$ can also be obtained by inverting and rewriting the sequence of assignments used in computation of $\acute{A}$.

$$\begin{aligned} x &:= x + y \\ y &:= \frac{1}{510783}\{714x + y\} \\ x &:= x - 716y \end{aligned}$$

$$\acute{A}^{-1} = \begin{pmatrix} -\frac{21}{24323} & -\frac{89}{39291} \\ \frac{34}{24323} & \frac{55}{39291} \end{pmatrix}$$

We can compute the A^{-1} by making some changes in the above sequence of assignments i.e., we will multiply the first column of $\acute{A}^{-1}$ by u and second by v. Therefore the sequence of assignments that can compute A^{-1} is as under:

$$\begin{aligned} x &:= u * x + v * y \\ y &:= \frac{1}{510783}\{714x + v * y\} \\ x &:= x - 716y \end{aligned}$$

The A^{-1} corresponding to matrix A is given below:

$$A^{-1} = \begin{pmatrix} -\frac{21}{1871} & \frac{89}{1871} \\ \frac{34}{1871} & -\frac{55}{1871} \end{pmatrix}$$

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

We give another example to explain the case, when the entries of the first column of matrix A are not Co prime

Example 9. *Consider the mapping E:* $(x, y) \longrightarrow (25x + 13y, 35x + 21y)$*, whose coefficients can be expressed by a matrix.*

$$A = \begin{pmatrix} m & n \\ p & q \end{pmatrix} = \begin{pmatrix} 25 & 13 \\ 35 & 21 \end{pmatrix}$$

Since $GCD(25, 35) = 5$*, Extract gcd from the first column, it will make the column co prime*

$$B = \begin{pmatrix} m & n \\ p & q \end{pmatrix} = \begin{pmatrix} 5 & 13 \\ 7 & 21 \end{pmatrix}$$

Now $GCD(5, 7) = 1$*, Therefore Applying Bézout's identity* $5u + 7v = 1$*, we can have a number of solutions, one of them is* $u = -4$*,* $v = 3$
Multiply first row by u *and second by* v*, we will get*

$$B = \begin{pmatrix} um & un \\ vp & vq \end{pmatrix} = \begin{pmatrix} -20 & -52 \\ 21 & 63 \end{pmatrix}$$

Solving the system of equations, we will have $a = 1, b = 11, c = 21, d = -168, e = 1, f = -1$ *such that the sequence of assignments for* $a = 1$*, is:*

$$\begin{aligned} x &:= x + 11y \\ y &:= -\frac{1}{4}(21x - 168y) \\ x &:= \frac{1}{3}(x - y) \end{aligned}$$

Now just replace $a = 5$*,The assignments for* $a = 5$*, are*

$$\begin{aligned} x &:= 5x + 11y \\ y &:= -\frac{1}{4}(21x - 168y) \\ x &:= \frac{1}{3}(x - y) \end{aligned}$$

2.5 Sequential Computation over Rings:

In this section, we explain how a linear mapping can be computed by a sequence of linear assignments over rings specifically over the ring $\mathbb{Z}/N\mathbb{Z}$. We begin with a lemma of Emmanuel Thomé, next we will explain a proposition in detail. We will also explain the idea of computing a mapping over $\mathbb{Z}/N\mathbb{Z}$ completely by giving different examples.

Lemma 2. *Let* $x_1, \cdots, x_n$ *be Co prime integers. Let* N *be an integer. There exists integers* $\lambda_2, \cdots, \lambda_n$ *such that* $x_1 + \Sigma_i \lambda_i x_i \in (\mathbb{Z}/N\mathbb{Z})^*$*, where* $(\mathbb{Z}/N\mathbb{Z})^*$ *denotes the group of invertible elements.*

Proof. Suppose that N is a prime power p^v. It is given that the integers x_i are Co prime, therefore there exist an integer i_0 (say) such that x_{i0} is Co prime to p. If x_1 is itself Co prime to p, then one can select $i_0 = 1$. If x_1 is not Co prime to p *i.e.* If x_1 is divisible by p, then $x_1 + x_{i0}$ is Co prime to p. So the result holds when $N = p^v$. For each prime power dividing N , we can therefore construct n integers $\lambda_1^{p^v}, \cdots, \lambda_n^{p^v}$ such that $\lambda_1^{p^v} = 1$ and $\Sigma_i(\lambda_i)^{p^v} x_i \in (\mathbb{Z}/p^v\mathbb{Z})^*$.
Using the Chinese Remainder Theorem, these vectors combine into a global solution $(1, \lambda_2, \cdots, \lambda_N)$ satisfying the required property. □

2.5.1 Assignment Matrices:

Assignment matrices have been used to prove the result. An assignment matrix is actually the modified form of identity matrix having a row different from identity matrix.

Definition 1. *A matrix A is said to be an assignment matrix, if there exist an integer i_0 such that for all row and column indices (i, j), one has either $i = i_0$ or $A_{i,j} = \delta_i^j$. So, if A is a square matrix then $A - I$ has at most one non-zero row, where I is the identity matrix.*

Example 10. *Consider a square matrix A as given below:*

$$\begin{pmatrix} 2 & 3 & 5 \\ 4 & 7 & 1 \\ 5 & 6 & 7 \end{pmatrix}$$

The given matrix can be decomposed into the following four assignment matrices under modulo 8 operation.

$$A_1 := \begin{pmatrix} 7 & 1 & 4 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} A_2 := \begin{pmatrix} 1 & 0 & 0 \\ 4 & 3 & 1 \\ 0 & 0 & 1 \end{pmatrix} A_3 := \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 7 & 1 & 2 \end{pmatrix} A_4 := \begin{pmatrix} 1 & 0 & 7 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Proposition 2. *Let N be an integer. Any $n \times n$ matrix over $(\mathbb{Z}/N\mathbb{Z})$ can be written as the product of at most $2n - 1$ assignment matrices.*

Proof. The result can be easily proved by the help of induction on the number of rows (representing by n) of the given matrix. Therefore if $n = 1$, the result is obvious.
For $n > 1$, we can proceed as follows. Consider the first column of the given matrix A (say). Suppose that the a_{11} is not an invertible element under modulo N. Let g represent the GCD of the first column *i.e.* $g = (a_{11}, a_{21}, \cdots, a_{n1})$. We can construct an invertible element using the lemma as given above. We apply this lemma to make a combination of the coefficients of this column equal to g times an invertible element of $(\mathbb{Z}/N\mathbb{Z})$, with the constraint that this combination has its first multiplier equal to 1. This implies that the $n \times n$ matrix T defined by $t_{i,j} = \delta_i^j$ for $i > 1$, and $t_{1,j} = \lambda_j$, where the multipliers $1, \lambda_2, \cdots, \lambda_n$ are obtained from the lemma above. Clearly the matrix T is an invertible assignment matrix, and the product TA has a coefficient at position $(1, 1)$ which is equal

to g times an invertible element modulo N. Now assuming that $a_{1,1} \in g(\mathbb{Z}/N\mathbb{Z})^*$. Let n' be the number of columns of A . Let $G = diag(g, 1, \cdots, 1)$, and let A' be the integer matrix AG^{-1} (A' has coefficients in $\mathbb{Z}$ because g is the GCD of the first column). We have $a'_{1,1} \in (\mathbb{Z}/N\mathbb{Z}^*)$. We form an $n' \times n'$ assignment matrix U defined by $u_{i,j} = \delta_i^j$ for $i > 1$, and $u_{1,j} = a'_{1,j}$. The matrix U is invertible modulo N (its determinant is $a'_{1,1}$). The first row of the matrix $A'' = A' * U^{-1}$ is equal to $(1, 0, \cdots, 0)$. Notice further that UG is an assignment matrix as well (even though not invertible modulo N). Putting together the different results we have that $A = T \times A'' \times (UG)$, where the matrix T may be omitted. Applying the result inductively on A'' completes the proof. □

2.5.2 Explanation and Construction:

For the given matrix A, if $a_{1,1}$ is invertible then matrix T will be the identity matrix of the same order as of matrix A and $A' = AG^{-1}$.
Otherwise matrix T will be defined by $t_{ij} = \delta_{ij} for i > 1 and t_{1,j} = \lambda_j$
i.e. the first row of the matrix T will consist of the multipliers $1, \lambda_1, \cdots, \lambda_n$, obtained by lemma given above. Matrix T may be a single matrix or $T = T_1, T_2, \cdots, T_n$ depending on number of prime factors of N.
An invertible matrix L can be constructed by solving the system $TL = A$.
i.e. $L_1 = T_1A, L_2 = T_2A, \cdots, L_n = T_nA$, and $A = (T_1 * T_2, \cdots, T_n)L_n$. g is the GCD of first column of matrix L_i obtained after T_i transformation. So if $A = (T_1 * T_2, \cdots, T_n)L_f$. Then $A' = L_fG^{-1}$, where L_f denote the final invertible matrix after $T_i, 1 \leq i \leq n$ transformation. Matrix A will finally satisfy the relation $A = (T_1 * T_2, \cdots, T_n)A''UG$.

Example 11. *Consider a mapping* $E : (x_1, x_2) \longrightarrow (2x_1 + 3x_2, 5x_1 + 7x_2)$. *The coefficients can be represented by a square matrix*

$$M = \begin{pmatrix} 2 & 3 \\ 5 & 7 \end{pmatrix}$$

Suppose that $N = 8 = 2^3$.
For $p^v = 2$, $(2, 2) \neq 1, \Longrightarrow m_{1,1}$ *is not invertible.*
Since $\lambda_2 = 1$, *therefore*

$$Matrix \quad T = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

Now, solving the system below

$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} 2 & 3 \\ 5 & 7 \end{pmatrix}$$

$a = -3 \, mod 8 = 5, b = -4 \, mod 8 = 4, c = 5, d = 7$
New matrix

$$M = \begin{pmatrix} 5 & 4 \\ 5 & 7 \end{pmatrix}$$

$$G = \begin{pmatrix} 5 & 0 \\ 0 & 1 \end{pmatrix} and\, G^{-1} = 5^{-1} \begin{pmatrix} 1 & 0 \\ 0 & 5 \end{pmatrix} mod\, 8 = \begin{pmatrix} 5 & 0 \\ 0 & 1 \end{pmatrix}$$

$$M' = MG^{-1} \begin{pmatrix} 5 & 4 \\ 5 & 7 \end{pmatrix} \begin{pmatrix} 5 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 25 & 4 \\ 25 & 7 \end{pmatrix} mod 8 = \begin{pmatrix} 1 & 4 \\ 1 & 7 \end{pmatrix}$$

$$U = \begin{pmatrix} 1 & 4 \\ 0 & 1 \end{pmatrix}, U^{-1} = \begin{pmatrix} 1 & -4 \\ 0 & 1 \end{pmatrix}$$

$$M'' = M'U^{-1} = \begin{pmatrix} 1 & 4 \\ 1 & 7 \end{pmatrix} \begin{pmatrix} 1 & -4 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1 & 3 \end{pmatrix}$$

$$UG = \begin{pmatrix} 1 & 4 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 5 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 5 & 4 \\ 0 & 1 \end{pmatrix}$$

$$TM''UG = \begin{pmatrix} 10 & 11 \\ 5 & 7 \end{pmatrix} mod 8 = \begin{pmatrix} 2 & 3 \\ 5 & 7 \end{pmatrix}$$

Hence the sequence linear assignments that can compute the given mapping is as follows:

$$a := 5a + 4b$$

$$b := a + 3b$$

$$a := a + b$$

Example 12. *Consider a mapping* $E : (x_1, x_2) \longrightarrow (2x_1 + 3x_2, 5x_1 + x_2)$. *The coefficients can be represented by a square matrix*

$$M = \begin{pmatrix} 2 & 3 \\ 5 & 1 \end{pmatrix} and \quad N = 6 = 2^1 * 3^1$$

$m_{1,1}$ *is not invertible with respect to* $p^v = 2$, $(2, 2) \neq 1$
$\lambda_1 = 1$, *and* $\lambda_2 = 1$,

$$T_1 = \begin{pmatrix} \lambda_1 & \lambda_2 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

Now,

$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} 2 & 3 \\ 5 & 1 \end{pmatrix}$$

$\Longrightarrow a = -3 mod 6 = 3, b = 2, c = 5, d = 1$
New matrix

$$L_1 = \begin{pmatrix} 3 & 2 \\ 5 & 1 \end{pmatrix}$$

Observe that $l_{1,1}$ *is not invertible, For* $p^v = 3$, $(3, 3) \neq 1$
$\lambda_1 = 1$, *and* $\lambda_2 = 1$,

$$T_2 = \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}$$

Now,

$$\begin{pmatrix}1 & 2\\ 0 & 1\end{pmatrix}\begin{pmatrix}a & b\\ c & d\end{pmatrix}=\begin{pmatrix}3 & 2\\ 5 & 1\end{pmatrix}$$

$\Longrightarrow a=-7\,mod\,6=5, b=0, c=5, d=1$
New matrix

$$L_2=\begin{pmatrix}5 & 0\\ 5 & 1\end{pmatrix}$$

$$G=\begin{pmatrix}5 & 0\\ 0 & 1\end{pmatrix}\ and G^{-1}=5^{-1}\begin{pmatrix}1 & 0\\ 0 & 5\end{pmatrix}\ mod\,6=5\begin{pmatrix}1 & 0\\ 0 & 5\end{pmatrix}=\begin{pmatrix}5 & 0\\ 0 & 1\end{pmatrix}$$

$$M'=MG^{-1}\begin{pmatrix}5 & 0\\ 5 & 1\end{pmatrix}\begin{pmatrix}5 & 0\\ 0 & 1\end{pmatrix}=\begin{pmatrix}25 & 0\\ 25 & 1\end{pmatrix}\ mod\,6=\begin{pmatrix}1 & 0\\ 1 & 1\end{pmatrix}$$

$$U=\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}, U^{-1}=\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}$$

$$M''=M'U^{-1}=\begin{pmatrix}1 & 0\\ 1 & 1\end{pmatrix}$$

$$UG=\begin{pmatrix}5 & 0\\ 0 & 1\end{pmatrix}$$

$$T_1T_2M''UG=\begin{pmatrix}20 & 3\\ 5 & 1\end{pmatrix}\ mod\,6=\begin{pmatrix}2 & 3\\ 5 & 1\end{pmatrix}$$

Hence the sequence of linear assignments that computes the given mapping is:

$$a := 5a$$
$$b := a + b$$
$$a := a + 3b$$

*

2.5.3 Computation of Inverse Mapping:

In this section, we will explain, by examples, how to compute inverse of a square matrix, by a sequence of assignments over the ring $\mathbb{Z}/N\mathbb{Z}$. Consequently, the method will lead to compute E^{-1}.

Example 13. *Consider a linear mapping*

$$E\,(x_1,\,x_2,\,x_3)\longrightarrow(2x_1+8x_2+6x_3,\,3x_1+13x_2+7x_3,\,5x_1+5x_2+x_3)$$

We are interested to compute a sequence of linear assignment under modulo 9.
Observe that, for a vector $X := (x_1, x_2, x_3)$*, the mapping* E *can be written as* $X := AX$*, where* A *is a matrix of coefficients. Therefore matrix* A *will be.*

$$A = \begin{pmatrix} 2 & 8 & 6 \\ 3 & 13 & 7 \\ 5 & 5 & 1 \end{pmatrix}$$

After performing modulo operation.

$$A = \begin{pmatrix} 2 & 8 & 6 \\ 3 & 4 & 7 \\ 5 & 5 & 1 \end{pmatrix}$$

The sequence of linear assignments that can compute mapping E*, under modulo* 9 *operation, is given by*

$$\begin{aligned} x_1 &:= 2x_1 + 8x_2 + 6x_3 \\ x_2 &:= 6x_1 + x_2 + 7x_3 \\ x_3 &:= 7x_1 + 3x_2 + x_3 \end{aligned}$$

Computing Inverse of Matrix A:

We can get a sequence of assignments that can compute E^{-1} *and ultimately* A^{-1}*. We obtain this sequence of assignments by inverting and rewriting the assignments (from bottom to top) that compute the mapping* E*.*

$$\begin{aligned} x_3 &:= x_3 - 7x_1 - 3x_2 \\ x_2 &:= x_2 - 6x_1 - 7x_3 \\ x_1 &:= 2^{-1}\,(x_1 - 8x_2 - 6x_3) \end{aligned}$$

In the last assignment, we will replace 2^{-1} *by* 5 *because* $2 * 5 = 1 \, mod \, 9$*, so that the last assignments becomes invertible modulo 9.*
Hence the sequence of assignments that can compute the inverse mapping under modulo 9 *is as given below:*

$$\begin{aligned} x_3 &:= x_3 - 7x_1 - 3x_2 \\ x_2 &:= x_2 - 6x_1 - 7x_3 \\ x_1 &:= 5\,(x_1 - 8x_2 - 6x_3) \end{aligned}$$

Consequently, we get the inverse of matrix A*.*

$$A^{-1} = \begin{pmatrix} 7 & 2 & 7 \\ 7 & 4 & 2 \\ 2 & 6 & 1 \end{pmatrix}$$

Verifying the product of assignment matrices:

The matrix

$$A = \begin{pmatrix} 2 & 8 & 6 \\ 3 & 13 & 7 \\ 5 & 5 & 1 \end{pmatrix}$$

is decomposed into three assignment matrices under modulo 9 *operation, these matrices are as given below:*

$$A_1 = \begin{pmatrix} 2 & 8 & 6 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} A_2 = \begin{pmatrix} 1 & 0 & 0 \\ 3 & 13 & 7 \\ 0 & 0 & 1 \end{pmatrix} A_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 5 & 5 & 1 \end{pmatrix}$$

Clearly the product of matrices $A_3 * A_2 * A_1 = A$.

2.5.4 Inverses mod k:

An integer a has an inverse mod k if and only if $GCD(a, k) = 1$. *It is not always true that a with* $a \not\equiv 0$ *(mod k) has an inverse mod k, e.g.* $2 \not\equiv 0$ *(mod* 4*). Since* $2 \times 2 = 4 \equiv 0$ *mod* 4. *Thus* 2 *has no inverse. Otherwise, we could multiply both sides of*

$$2 \times 2 \equiv 0 \; mod \; 4$$

by the inverse of 2 *and get the false result* $2 \equiv 0$ *mod* 4. *If p is a prime number, then for each* $a \not\equiv 0$ *mod p has a multiplicative inverse mod p.*

Existence of Inverse of Matrix A:

Consider again the mapping

$$E\,(x_1,\, x_2,\, x_3) \longrightarrow (2x_1 + 8x_2 + 6x_3,\, 3x_1 + 13x_2 + 7x_3,\, 5x_1 + 5x_2 + x_3)$$

as given in Example 2. This mapping can also be computed by the following sequence of assignments under modulo 12.

$$\begin{aligned} x_1 &:= 5x_1 + 5x_2 + 9x_3 \\ x_2 &:= 3x_1 + 10x_2 + 4x_3 \\ x_3 &:= x_1 + 4x_3 \\ x_1 &:= x_1 + 3x_2 \end{aligned}$$

By inverting and rewriting this sequence of assignments, we get the following sequence of assignments.

$$\begin{aligned} x_1 &:= x_1 - 3x_2 \\ x_3 &:= 4^{-1}\,(x_3 - x_1) \\ x_2 &:= 10^{-1}\,(x_2 - 3x_1 - 4x_3) \\ x_1 &:= 5^{-1}\,(x_1 - 5x_2 - 9x_3) \end{aligned}$$

Observe that, the second and third assignments are not invertible because the integers 4 *and* 10 *have not their inverses under modulo* 12 *operation. Moreover, the Determinant of matrix* A

$$|A| = \begin{vmatrix} 2 & 8 & 6 \\ 3 & 13 & 7 \\ 5 & 5 & 1 \end{vmatrix} = 248$$

is not invertible modulo 12*. Thus* A^{-1} *modulo* 12 *is not computable by the sequence of assignments as given above. Hence, to compute inverse mapping the linear assignments should must be invertible.*

Chapter 3

In Situ Computation of Linear Mappings over $\mathbb{Z}$:

In this chapter, we describe In Situ computation of linear mapping by a sequence of linear assignments over the set of integers $\mathbb{Z}$. We start in proving a proposition and then we explain the algorithm completely by giving different examples.

Proposition 3. *Let E be a linear mapping on Z^n, where Z is the set of integers and n is a positive integer. There exists a finite sequence $f_1, f_2, \cdots f_n, g_{n-1}, \cdots, g_2, g_1$ of linear mappings from Z^n to Z such that the program :*

$$
\begin{aligned}
&x_1 := f_1(x_1, x_2, \cdots, x_n)\\
&x_2 := f_2(x_1, x_2, \cdots, x_n)\\
&\vdots := \quad\ \vdots \quad\ \vdots\\
&x_n := f_n(x_1, x_2, \cdots, x_n)\\
&x_{n-1} := g_{n-1}(x_1, x_2, \cdots, x_n)\\
&\vdots := \quad\ \vdots \quad\ \vdots\\
&x_2 := g_2(x_1, x_2, \cdots, x_n)\\
&x_1 := g_1(x_1, x_2, \cdots, x_n)
\end{aligned}
$$

performs the operation $X := E(X)$ for any $X = (x_1, x_2, \cdots, x_n)$ in Z^n. The sequence $g_n, g_{n-1}, \cdots, g_2, g_1$ contains exactly n number of assignments.

Proof. The mapping $E := E(X)$ can be considered as $X := AX$, where A is a matrix of coefficients *i.e.* $a_{ij} \in \mathbb{Z}$ for $i, j = 1, 2, \cdots, n$. Each row of the matrix AX represents

a component of the mapping E at current values of the variables $(x_1, x_2, \cdots, x_n)$ and is considered to be a linear mapping . For example, the jth row $a_{j1} + a_{j2} + a_{j3} + \ldots + a_{jn}$, of matrix AX will be the E_j component of mapping E.

Case 1: If $a_{ij} < 0$, $for\, j > i$, then multiply jth column by -1, so that $a_{ij} > 0$ and x_j will be modified to $x_j := -x_j$. Similarly, if $a_{ii} < 0$, then ith column will be multiplied by -1, so that $a_{ii} > 0$ and x_i will be modified to $x_i := -x_i$.

Case 2: If $a_{ii} > a_{ij}$, $for\, j > i$, then the program will perform the operation $C_i := C_i - C_j$, and x_j will be modified to $x_j := \alpha x_j + \Delta$, where Δ does not depend on x_j. C_i denotes the ith column of matrix A.

Case 3: If $a_{ij} \geq a_{ii}$, $for\, j > i$, then the program will perform the operation $C_j := C_j - C_i$, and x_i will be modified to $x_i := \alpha x_i + \Delta$, where Δ does not depend on x_i.

Case 4: If $a_{ii} = 0$, then the program will perform the operation $C_i := C_i - C_j$, and x_j will be modified to $x_j := \alpha x_j + \Delta$, where Δ does not depend on x_j.

The whole procedure works iteratively until the matrix A modified to be a triangular matrix. After the completion of this procedure the matrix A will become a lower triangular matrix L (say) as given below.

$$L := \begin{pmatrix} l_{11} & 0 & 0 & \ldots & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & \ldots & 0 & 0 & 0 \\ l_{31} & l_{32} & l_{33} & \ldots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ldots & \vdots & \vdots & \\ l_{n-2,1} & l_{n-2,2} & l_{n-2,3} & \ldots & l_{n-2,n-2} & 0 & 0 \\ l_{n-1,1} & l_{n-1,2} & l_{n-1,3} & \ldots & l_{n-1,n-2} & l_{n-1,n-1} & 0 \\ l_{n1} & l_{n2} & l_{n3} & \ldots & l_{n,n-2} & l_{n,n-1} & l_{nn} \end{pmatrix}$$

The last n assignments can be obtained directly from the matrix L and that will be of the form.

$$x_n := l_{n1}x_1 + l_{n2}x_2 + l_{n3}x_3 + \ldots + l_{n,n-2}x_{n-2} + l_{n,n-1}x_{n-1} + l_{nn}x_n$$

$$x_{n-1} := l_{n-1,1}x_1 + l_{n-1,2}x_2 + l_{n-1,3}x_3 + \ldots + l_{n-1,n-2}x_{n-2} + l_{n-1,n-1}x_{n-1}$$

$$x_{n-2} := l_{n-2,1}x_1 + l_{n-2,2}x_2 + l_{n-2,3}x_3 + \ldots + l_{n-2,n-2}x_{n-2}$$

$$\ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots$$

$$\ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots$$

$$\ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots$$

$$x_3 := l_{31}x_1 + l_{32}x_2 + l_{33}x_3$$

$$x_2 := l_{21}x_1 + l_{22}x_2$$

$$x_1 := l_{11}x_1$$

Let $\acute{A}$ and $\acute{V}$ denote the modified form of the matrix A and vector V after performing any operation. After each operation it should must preserve and compute the original mapping effectively *i.e.*

$$MV = \acute{M}\acute{V}$$

If T denotes the transformation matrix then

$$\begin{aligned}
&\acute{M} = MT \\
&\acute{V} = T^{-1}V \\
&\implies \acute{M}\acute{V} = M(TT^{-1})V = MV
\end{aligned}$$

Observe that, to compute mapping E the last n assignments are not necessarily to be invertible, because in this last sequence of assignments, the reference of initial assignments are not required to be used for the next assignments.
The similar linear assignments can be combined to a single linear assignment that have the same effect.
In general, if the assignment $x_i := \alpha x_i + \Delta$ repeated k times then we can write it as a single assignment of the form

$$x_i := \alpha^k x_i + \Delta\left(\alpha^{k-1} + \alpha^{k-2} + \ldots + \alpha^2 + \alpha^1 + \alpha^0\right)$$

We can modify the algorithm to avoid from repeated similar assignments. □

3.0.5 Explanation and Construction:

Consider a linear mapping

$$E\left(x_1,\, x_2,\, x_3\right) \longrightarrow \left(5x_1 - 3x_2 + 5x_3,\, 3x_1 - 7x_3,\, 4x_1 + 8x_2 + 13x_3\right)$$

We are interested to compute the given mapping by a sequence of linear assignments such that the coefficients of these assignments are integers. The coefficients of the given mapping E can be taken as a square matrix A:

$$A := \begin{pmatrix} 5 & -3 & 5 \\ 3 & 0 & -7 \\ 4 & 8 & 13 \end{pmatrix}$$

We shall compute the mapping E by applying the procedure, keeping in mind different cases, as described above in the proposition 2. Since $a_{12} < 0$, therefore, we perform the operation $C_2 := -C_2$ because we are in the first case. The linear assignment corresponding to this operation is $x_2 := -x_2$ and the matrix A will be modified to the form

$$A := \begin{pmatrix} 5 & 3 & 5 \\ 3 & 0 & -7 \\ 4 & -8 & 13 \end{pmatrix}$$

Since $a_{11} = 5 > a_{12} = 3$, therefore, we perform the operation $C_1 := C_1 - C_2$ because we are in the second case. The linear assignment corresponding to this operation is $x_2 := x_1 + x_2$ and the matrix A will be modified.

$$A := \begin{pmatrix} 2 & 3 & 5 \\ 3 & 0 & -7 \\ 12 & -8 & 13 \end{pmatrix}$$

Now, $a_{12} = 3 \geq a_{11} = 2$ therefore, we are in the third case and we perform the operation $C_2 := C_2 - C_1$. The linear assignment corresponding to this operation is $x_1 := x_1 + x_2$ and the matrix A will be modified.

$$A := \begin{pmatrix} 2 & 1 & 5 \\ 3 & -3 & -7 \\ 12 & -20 & 13 \end{pmatrix}$$

Since $a_{11} = 2 > a_{12} = 1$, therefore, we perform the operation $C_1 := C_1 - C_2$. The linear assignment corresponding to this operation is $x_2 := x_1 + x_2$ and the matrix A will be modified.

$$A := \begin{pmatrix} 1 & 1 & 5 \\ 6 & -3 & -7 \\ 32 & -20 & 13 \end{pmatrix}$$

Since $a_{12} = 1 \geq a_{11} = 1$, therefore, we perform the operation $C_2 := C_2 - C_1$. The linear assignment corresponding to this operation is $x_1 := x_1 + x_2$ and the matrix A will be modified.

$$A := \begin{pmatrix} 1 & 0 & 5 \\ 6 & -9 & -7 \\ 32 & -52 & 13 \end{pmatrix}$$

Observe that a_{12} became zero, We will continue this process until $a_{13} = 0$.
Since $a_{13} = 5 \geq a_{11} = 1$, we perform the operation $C_3 := C_3 - C_1$. The linear assignment corresponding to this operation is $x_1 := x_1 + x_3$. The matrix A will be modified.

$$A := \begin{pmatrix} 1 & 0 & 4 \\ 6 & -9 & -13 \\ 32 & -52 & -19 \end{pmatrix}$$

Since $a_{13} = 4 \geq a_{11} = 1$, we perform the operation $C_3 := C_3 - C_1$. The linear assignment corresponding to this operation is $x_1 := x_1 + x_3$. The matrix A will be modified.

$$A := \begin{pmatrix} 1 & 0 & 3 \\ 6 & -9 & -19 \\ 32 & -52 & -51 \end{pmatrix}$$

Since $a_{13} = 3 \geq a_{11} = 1$, we perform the operation $C_3 := C_3 - C_1$. The linear assignment corresponding to this operation is $x_1 := x_1 + x_3$. The matrix A will be modified.

$$A := \begin{pmatrix} 1 & 0 & 2 \\ 6 & -9 & -25 \\ 32 & -52 & -83 \end{pmatrix}$$

Since $a_{13} = 2 \geq a_{11} = 1$, we perform the operation $C_3 := C_3 - C_1$. The linear assignment corresponding to this operation is $x_1 := x_1 + x_3$. The matrix A will be modified.

$$A := \begin{pmatrix} 1 & 0 & 1 \\ 6 & -9 & -31 \\ 32 & -52 & -115 \end{pmatrix}$$

Since $a_{13} = 1 \geq a_{11} = 1$, we perform the operation $C_3 := C_3 - C_1$. The linear assignment corresponding to this operation is $x_1 := x_1 + x_3$. The matrix A will be modified.

$$A := \begin{pmatrix} 1 & 0 & 0 \\ 6 & -9 & -37 \\ 32 & -52 & -147 \end{pmatrix}$$

Now the first column will remain unchanged throughout the next operations, we will focus on second diagonal entry.
Since $a_{23} = -37 < 0$, we perform the operation $C_3 := -C_3$. The linear assignment corresponding to this operation is $x_3 := -x_3$. The matrix A will be modified.

$$A := \begin{pmatrix} 1 & 0 & 0 \\ 6 & -9 & 37 \\ 32 & -52 & 147 \end{pmatrix}$$

Since $a_{22} = -9 < 0$, we perform the operation $C_2 := -C_2$. The linear assignment corresponding to this operation is $x_2 := -x_2$. The matrix A will be modified.

$$A := \begin{pmatrix} 1 & 0 & 0 \\ 6 & 9 & 37 \\ 32 & 52 & 147 \end{pmatrix}$$

Since $a_{23} = 37 \geq a_{22} = 9$, we perform the operation $C_3 := C_3 - C_2$. The linear assignment corresponding to this operation is $x_2 := x_2 + x_3$. The matrix A will be modified.

$$A := \begin{pmatrix} 1 & 0 & 0 \\ 6 & 9 & 28 \\ 32 & 52 & 95 \end{pmatrix}$$

since $a_{23} = 28 \geq a_{22} = 9$, we perform the operation $C_3 := C_3 - C_2$. The linear assignment corresponding to this operation is $x_2 := x_2 + x_3$. The matrix A will be modified.

$$A := \begin{pmatrix} 1 & 0 & 0 \\ 6 & 9 & 19 \\ 32 & 52 & 43 \end{pmatrix}$$

Since $a_{23} = 19 \geq a_{22} = 9$, we perform the operation $C_3 := C_3 - C_2$. The linear assignment corresponding to this operation is $x_2 := x_2 + x_3$. The matrix A will be modified.

$$A := \begin{pmatrix} 1 & 0 & 0 \\ 6 & 9 & 10 \\ 32 & 52 & -9 \end{pmatrix}$$

Since $a_{23} = 10 \geq a_{22} = 9$, we perform the operation $C_3 := C_3 - C_2$. The linear assignment corresponding to this operation is $x_2 := x_2 + x_3$. The matrix A will be modified.

$$A := \begin{pmatrix} 1 & 0 & 0 \\ 6 & 9 & 1 \\ 32 & 52 & -61 \end{pmatrix}$$

Since $a_{22} = 9 > a_{23} = 1$, we perform the operation $C_2 := C_2 - C_3$. The linear assignment corresponding to this operation is $x_3 := x_2 + x_3$. The matrix A will be modified.

$$A := \begin{pmatrix} 1 & 0 & 0 \\ 6 & 8 & 1 \\ 32 & 113 & -61 \end{pmatrix}$$

Since $a_{22} = 8 > a_{23} = 1$, we perform the operation $C_2 := C_2 - C_3$. The linear assignment corresponding to this operation is $x_3 := x_2 + x_3$. The matrix A will be modified.

$$A := \begin{pmatrix} 1 & 0 & 0 \\ 6 & 7 & 1 \\ 32 & 174 & -61 \end{pmatrix}$$

Since $a_{22} = 7 > a_{23} = 1$, we perform the operation $C_2 := C_2 - C_3$. The linear assignment corresponding to this operation is $x_3 := x_2 + x_3$. The matrix A will be modified.

$$A := \begin{pmatrix} 1 & 0 & 0 \\ 6 & 6 & 1 \\ 32 & 235 & -61 \end{pmatrix}$$

Since $a_{22} = 6 > a_{23} = 1$, we perform the operation $C_2 := C_2 - C_3$. The linear assignment corresponding to this operation is $x_3 := x_2 + x_3$. The matrix A will be modified.

$$A := \begin{pmatrix} 1 & 0 & 0 \\ 6 & 5 & 1 \\ 32 & 296 & -61 \end{pmatrix}$$

Since $a_{22} = 5 > a_{23} = 1$, we perform the operation $C_2 := C_2 - C_3$. The linear assignment corresponding to this operation is $x_3 := x_2 + x_3$. The matrix A will be modified.

$$A := \begin{pmatrix} 1 & 0 & 0 \\ 6 & 4 & 1 \\ 32 & 357 & -61 \end{pmatrix}$$

Since $a_{22} = 4 > a_{23} = 1$, we perform the operation $C_2 := C_2 - C_3$. The linear assignment corresponding to this operation is $x_3 := x_2 + x_3$. The matrix A will be modified.

$$A := \begin{pmatrix} 1 & 0 & 0 \\ 6 & 3 & 1 \\ 32 & 418 & -61 \end{pmatrix}$$

Since $a_{22} = 3 > a_{23} = 1$, we perform the operation $C_2 := C_2 - C_3$. The linear assignment corresponding to this operation is $x_3 := x_2 + x_3$. The matrix A will be modified.

$$A := \begin{pmatrix} 1 & 0 & 0 \\ 6 & 2 & 1 \\ 32 & 479 & -61 \end{pmatrix}$$

Since $a_{22} = 2 > a_{23} = 1$, we perform the operation $C_2 := C_2 - C_3$. The linear assignment corresponding to this operation is $x_3 := x_2 + x_3$. The matrix A will be modified.

$$A := \begin{pmatrix} 1 & 0 & 0 \\ 6 & 1 & 1 \\ 32 & 540 & -61 \end{pmatrix}$$

Since $a_{23} = 1 \geq a_{22} = 1$, we perform the operation $C_3 := C_3 - C_2$. The linear assignment corresponding to this operation is $x_2 := x_2 + x_3$. The matrix A will be modified.

$$A := \begin{pmatrix} 1 & 0 & 0 \\ 6 & 1 & 0 \\ 32 & 540 & -601 \end{pmatrix}$$

The matrix A converted into a lower triangular matrix. The last three assignments will be written directly from this lower triangular matrix. These three assignments are: $x_3 := 32x_1 + 540x_2 - 601x_3$, $x_2 := 6x_1 + x$ and $x_1 := x_1$
Hence the given mapping E can be computed by the sequence of assignments as given below:

$$x_2 = -x_2$$
$$x_2 = x_1 + x_2$$
$$x_1 = x_1 + x_2$$
$$x_2 = x_1 + x_2$$
$$x_1 = x_1 + x_2$$
$$x_1 = x_1 + x_3$$
$$x_1 = x_1 + x_3$$
$$x_1 = x_1 + x_3$$

$$x_1 = x_1 + x_3$$
$$x_1 = x_1 + x_3$$
$$x_3 = -x_3$$
$$x_2 = -x_2$$
$$x_2 = x_2 + x_3$$
$$x_2 = x_2 + x_3$$
$$x_2 = x_2 + x_3$$
$$x_2 = x_2 + x_3$$
$$x_3 = x_2 + x_3$$
$$x_3 = x_2 + x_3$$
$$x_3 = x_2 + x_3$$
$$x_3 = x_2 + x_3$$
$$x_3 = x_2 + x_3$$
$$x_3 = x_2 + x_3$$
$$x_3 = x_2 + x_3$$
$$x_3 = x_2 + x_3$$
$$x_2 = x_2 + x_3$$
$$x_3 = 32 * x_1 + 540 * x_2 - 601 * x_3$$
$$x_2 = 6 * x_1 + x_2$$
$$x_1 = x_1$$

The total number of assignments is 28, and can be reduced to 14 by combining consecutive assignments of the same variable.

$$x_2 = -x_2$$
$$x_2 = x_1 + x_2$$
$$x_1 = x_1 + x_2$$
$$x_2 = x_1 + x_2$$
$$x_1 = x_1 + x_2$$
$$x_1 = x_1 + 5x_3$$
$$x_3 = -x_3$$
$$x_2 = -x_2$$

$$x_2 = x_2 + 4x_3$$
$$x_3 = x_2 + 8x_3$$
$$x_2 = x_2 + x_3$$
$$x_3 = 32x_1 + 540x_2 - 601x_3$$
$$x_2 = 6x_1 + x_2$$
$$x_1 = x_1$$

3.0.6 Computing Inverse mapping over $\mathbb{Z}$:

We can compute the inverse mapping E^{-1} over $\mathbb{Z}$ by inverting each assignment in the sequence (that can compute mapping E) and rewriting from bottom to top. We explain by the following example

Example 14. *Suppose that E be a linear mapping and is defined as*

$$E\,(x_1,\, x_2) \longrightarrow (8x_1 + 13x_2,\, 13x_1 + 21x_2)$$

The matrix of coefficients A is

$$A := \begin{pmatrix} 8 & 13 \\ 13 & 21 \end{pmatrix}$$

The given mapping E can be computed by the following sequence of assignments

$$x_1 := -x_1 - 2x2$$
$$x_2 := 3x_1 + x2$$
$$x_1 := x_1 - 3x2$$
$$x_2 := 2x_1 + x2$$

Now, to compute E^{-1}, invert each assignment and rewrite from bottom to top. We get the following sequence of linear assignments that can compute E^{-1}.

$$x_2 := x_2 - 2x1$$
$$x_1 := x_1 + 3x2$$
$$x_2 := x_2 - 3x1$$
$$x_1 := -x_1 - 2x2$$

Consequently, we can compute A^{-1}, i.e.

$$A^{-1} := \begin{pmatrix} -21 & 13 \\ 13 & -8 \end{pmatrix}$$

Existence of Inverse matrix A^{-1}:

The computation of inverse mapping is not always possible over $\mathbb{Z}$ by a sequence of linear assignments. Consider the following example

Example 15. *Consider a mapping E*

$$E\,(x_1,\, x_2,\, x_3) \longrightarrow (2x_1 + 3x_2 + 5x_3,\, 3x_1 + 4x_2 - 7x_3,\, 8x_2 + 13x_3)$$

The mapping E can be computed by the following sequence of assignments.

$$x_1 = x_1 + x_2$$
$$x_2 = 2x_1 + x_2$$
$$x_2 = x_1 + x_2$$
$$x_1 = -x_1$$
$$x_1 = x_1 + x_2$$
$$x_1 = x_1 + 5x_3$$
$$x_3 = -x_3$$
$$x_2 = x_2 + 7x_3$$
$$x_3 = 24 * x_1 - 16 * x_2 + 219 * x_3$$
$$x_2 = x_2$$
$$x_1 = x_1$$

The matrix of coefficients A for the mapping E can be written as

$$A := \begin{pmatrix} 2 & 3 & 5 \\ 3 & 4 & -7 \\ 0 & 8 & 13 \end{pmatrix}$$

and

$$A^{-1} := \frac{1}{219} \begin{pmatrix} 108 & 1 & -41 \\ -39 & 26 & 29 \\ 24 & -16 & -1 \end{pmatrix}$$

But $\dfrac{1}{219} \notin \mathbb{Z}$. Observe that the assignment

$$x_3 = 24 * x_1 - 16 * x_2 + 219 * x_3$$

is not invertible. Therefore the inverse mapping E^{-1} can be computed by inverting and rewriting the above sequence of assignments. Moreover E^{-1} is computable by the sequence of assignments iff E is a bijective mapping. Hence the given mapping E is not a bijective mapping.

3.0.7 Investigating Bounds for the number of assignments:

We are interested to find the minimum number of assignments required to compute a given linear mapping over $\mathbb{Z}$. We proceed by developing different relations between mappings that can help to find the minimum number of assignments and we investigate through finding different counter examples. We are giving a proof below that shows that six assignments are not sufficient to compute any linear mapping over $\mathbb{Z}^2$.

Lemma 3. *It is not possible that every linear mapping*

$$E : (x,\, y) \longrightarrow (mx + ny,\, px + qy)$$

, where $m,\ n,\ p,\ q \in \mathbb{Z}$, can be computed by a sequence of at most 6 linear assignments

$$\begin{aligned} x &:= ax + by \\ y &:= cx + dy \\ x &:= ex + fy \\ y &:= gx + hy \\ x &:= ix + jy \\ y &:= kx + ly \end{aligned}$$

where $a,\ b,\ c,\ d,\ e,\ f,\ g,\ h,\ i,\ j,\ k,$ and $l \in \mathbb{Z}$,

Proof. Consider a linear mapping $E(x, y) \longrightarrow (461x + 286y, 353x + 219y)$, that can be viewed as a matrix of order 2×2 as given below.

$$\begin{pmatrix} 461 & 286 \\ 353 & 219 \end{pmatrix} = A(say)$$

We can establish a system of four equations(by evaluating assignments) as given below:

$$\begin{aligned} 461 &= a(ie + ifc + jge + jgfc + jhc) \\ 286 &= ieb + ifcb + ifd + jgeb + jgfcb + jgfd + jhcb + jhd \\ 353 &= 461k + lgea + lgfca + lhca \\ 219 &= 286k + lgeb + lgfcb + lgfd + lhcb + lhd \end{aligned} \tag{2}$$

$Determinant(A) = mq - pn = aedihl$
$\Longrightarrow$ The product of six variables $a,\ d,\ e,\ h,\ i,$ and l should must be equal to the determinant of matrix A.
Since, the determinant of matrix A is equal to 1. Therefore, $a * e * d * h * i * l = 1$ and each of six variables can take only the value $\{1, -1\}$.
There are $P(6, 2) + 2 = 32$, cases of assigning values to $a,\ d,\ e,\ h,\ i,$ and l.
Let us consider one case that is $a = 1,\ e = 1,\ d = 1,\ h = 1,\ i = 1,\ l = 1$.

We will have equations:

$$\begin{aligned} 460 &= fc + jg + jgfc + jc \\ 286 &= 461b + f + jgf + j \\ 353 &= 461k + g + gfc + c \\ 218 &= 286k + 353b - 461bk + gf \end{aligned} \tag{3}$$

There are three possible general solutions for system (3) of equations:
The first possible solution is of the form:

$$c = \frac{461}{j}, j = j, f = f, b = -\frac{1}{461}j + \frac{286}{461}$$
$$k = \frac{1}{461}\frac{-460j + 461f + 353j^2}{j^2}, g = -\frac{1}{j}$$

The second possible solution is of the form:

$$j = j, g = \frac{460}{j}, k = \frac{1}{461}\frac{-460 + 353j}{j}$$
$$b = -f - \frac{1}{461}j + \frac{286}{461}, f = f, c = 0$$

The third possible solution is of the form:

$$j = j, g = g, k = -\frac{1}{461}\frac{461g + c - 353 - 353jg}{1 + jg}$$
$$c = c, b = \frac{1}{461}\frac{jg - 460 + 286c}{c}, f = -\frac{-460 + jg + jc}{c(1 + jg)}$$

Consider the first possible solution:

$$\text{Suppose that } g = -\frac{1}{j} \in \mathbb{Z} \implies j = 1 \; or \; -1$$

Now, If $j = 1$ then

$$g = -1 \in \mathbb{Z}, \text{ but } b = -\frac{1}{461} + \frac{286}{461} = \frac{285}{461} \notin \mathbb{Z}$$

Similarly, If $j = -1$ then

$$g = 1 \in \mathbb{Z}, \text{ but } b = \frac{1}{461} + \frac{286}{461} = \frac{287}{461} \notin \mathbb{Z}$$

Therefore integer solution is not possible in the first solution.
In the second possible solution:

$$g = \frac{460}{j}, k = \frac{1}{461}\frac{-460 + 353j}{j}$$

$$\text{k can be written as } k = \frac{-g + 353}{461}$$

Observe that if $k \in \mathbb{Z}$ then $-g + 353 \geq 461$

$$\implies g \leq 353 - 461 \implies g \leq -108$$

But, since g is a divisor of 460, and, g can take only the value $\{-115, -230, -460\}$ because $g \leq -108$. But for any of these values $\{-115, -230, -460\}$ $k \notin \mathbb{Z}$.
Notice that,

$$if\ g = -230\ then\ k = \frac{230 + 353}{461} = \frac{583}{461} \notin \mathbb{Z}$$

$$if\ g = -115\ then\ k = \frac{115 + 353}{461} = \frac{468}{461} \notin \mathbb{Z}$$

$$if\ g = -460\ then\ k = \frac{460 + 353}{461} = \frac{813}{461} \notin \mathbb{Z}$$

Therefore integer solution is not possible using second possible solution.
Now, in the third possible solution:

$$b = \frac{1}{461}\frac{jg - 460 + 286c}{c} \text{ and } f = -\frac{-460 + jg + jc}{c(1 + jg)}$$

Since c divides both $jg - 460 + 286c$ and $-460 + jg + jc$, therefore, if $b, f \in \mathbb{Z}$ then there should exist $k_1, k_2 \in \mathbb{Z}$ such that

$$460 - jg - jc = ck_1 \tag{3.1}$$

and

$$460 - jg = c(286 - 461k_2) \tag{3.2}$$

By (3.1) and (3.2)

$$c(j + k_1) = c(286 - 461k_2)$$

$$\implies j + k_1 = 286 - 461k_2$$

$$\implies k_2 = -\frac{-286 + k_1 + j}{461}$$

If $k_2 \in \mathbb{Z}$ then $-286 + k_1 + j \geq 461$

$$\implies k_1 + j \geq 747 \tag{3.3}$$

$$\text{But form equation (3.1) } c = \frac{460 - jg}{j + k_1}$$

and if $c \in \mathbb{Z}$ then

$$j + k_1 \leq 460 - jg \tag{3.4}$$

By (3.3) and (3.4), we can write

$$460 - jg = 747 \Longrightarrow jg = -287$$

$\Longrightarrow$ For j and g to be integers, they should must be factors of 287 and

$$\{\pm 1,\ \pm 7,\ \pm 41,\ \pm 287\}$$

are the possible factors of 287.
Let, for example, $g = -287$ then $j = 1$

$$\Longrightarrow k_1 = 746 \Longrightarrow k_2 = -1 \Longrightarrow c = 1$$

$$But,\ then,\ f = -\frac{-460 - 287 + 1}{1 - 287} = -\frac{34}{13} \notin \mathbb{Z}$$

For all divisors of 287, $f \notin \mathbb{Z}$.
The other 31 possibilities of assigning values $\{1, -1\}$ to a, d, e, h, i, and l also do not work and can be proved by the similar arguments. □

We have established a useful relation between different mappings with coefficients taken from Fibonacci sequence. Next, we explain a proof related to these relations.

Definition 2. *We will define a Fibonacci-like sequence as*

$$F_n = F_{n-1} + F_{n-2}$$

where $F_0 = F_1 = 1$.

Definition 3. *We will define a Fibonancci-like matrix to be a matrix in the form*

$$\begin{pmatrix} F_{n-1} & F_n \\ F_n & F_{n+1} \end{pmatrix}$$

We define also the relations

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^{n+1} = \begin{pmatrix} F_{n-1} & F_n \\ F_n & F_{n+1} \end{pmatrix}$$

$$\begin{pmatrix} F_2 & F_3 \\ F_3 & F_4 \end{pmatrix} * \begin{pmatrix} F_{4n+1} & F_{4n+2} \\ F_{4n+2} & F_{4n+3} \end{pmatrix} = \begin{pmatrix} F_{4n+4} & F_{4n+5} \\ F_{4n+5} & F_{4n+6} \end{pmatrix}$$

Lemma 4. *Let* $E_n : (x,\ y) \longrightarrow (F_{n-1}x + F_n y,\ F_n x + F_{n+1} y)$ *be the mapping on* $\mathbb{Z}^2$, *where* F_n *is the Fibonacci number. The mapping* E_{4k+2} *can be computed with* $2k + 2$ *number of assignments, where* $k = 0,\ 1,\ 2, \cdots, n$.

Proof. For $n = 2$, $E_2 = (F_1x + F_2y, F_2x + F_3y)$ can be expressed as

$$A_2 = \begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix}$$

$$\text{The assignments for } A_2 = \begin{cases} x := x + 2y \\ y := 2x - y \end{cases}$$

$$A_3 = \begin{pmatrix} 2 & 3 \\ 3 & 5 \end{pmatrix}$$

$$\text{The assignments for } A_3 = \begin{cases} x := x + 2y \\ y := 3x - y \\ x := -x + y \end{cases}$$

Observe that

$$A_3 = \begin{pmatrix} 2 & 3 \\ 3 & 5 \end{pmatrix} * A_2 = \begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix} = A_6 = \begin{pmatrix} 8 & 13 \\ 13 & 21 \end{pmatrix}$$

$$\text{The assignments for } A_6 = \begin{cases} x := x + 2y \\ y := 3x - y \\ x := -x + y \\ x := x + 2y \\ y := 2x - y \end{cases} = \begin{cases} x := x + 2y \\ y := 3x - y \\ x := -x + 3y \\ y := 2x - y \end{cases}$$

$$\text{Similarly, } A_3 = \begin{pmatrix} 2 & 3 \\ 3 & 5 \end{pmatrix} * A_6 = \begin{pmatrix} 8 & 13 \\ 13 & 21 \end{pmatrix} = A_{10} = \begin{pmatrix} 55 & 89 \\ 89 & 144 \end{pmatrix}$$

$$\text{The assignments for } A_{10} = \begin{cases} x := x + 2y \\ y := 3x - y \\ x := -x + y \\ x := x + 2y \\ y := 3x - y \\ x := -x + 3y \\ y := 2x - y \end{cases} = \begin{cases} x := x + 2y \\ y := 3x - y \\ x := -x + 3y \\ y := 3x - y \\ x := -x + 3y \\ y := 2x - y \end{cases}$$

In general, $A_3 * A_{4k+2} = A_{4k+6}$

$$\text{The assignments for } A_{4k+6} = \begin{cases} x := x + 2y \\ y := 3x - y \\ x := -x + y \\ x := x + 2y \\ y := 3x - y \\ x := -x + 3y \\ \vdots \\ y := 3x - y \\ x := -x + 3y \\ y := 2x - y \end{cases} = \begin{cases} x := x + 2y \\ y := 3x - y \\ x := -x + 3y \\ \vdots \\ y := 3x - y \\ x := -x + 3y \\ y := 2x - y \end{cases}$$

The number of assignments for A_{4k+2} is equal to $2k+2$.

□

Proposition 4. *The mapping* $E : \mathbb{Z}^m \longrightarrow \mathbb{Z}^m$, $m > 2$

$$E : (x_1, x_2, x_3, \cdots, x_n) = \begin{pmatrix} F_1x_1 + F_2x_2 + F_3x_3 + \ldots + F_mx_n \\ F_2x_1 + F_3x_2 + F_4x_3 + \ldots + F_{m+1}x_n \\ F_3x_1 + F_4x_2 + F_5x_3 + \ldots + F_{m+2}x_n \\ \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \\ F_nx_1 + F_{n+1}x_2 + F_{n+2}x_3 + \ldots + F_{2n-1}x_n \end{pmatrix}$$

such that $F_n := F_{n-1} + F_{n-2}, \forall n \in \mathbb{Z}$*, can be computed with* $m+2$ *number of linear assignments.*

Proof. The given mapping E can be imagined as $X := AX$ where A is a matrix such that whose entries satisfy the relation $F_n := F_{n-1} + F_{n-2}, \forall n \in \mathbb{Z}$. we construct the first two mappings as given below:

$$x_1 := x_1 + x_2 + 2x_3 + 3x_4 + 5x_5 + \ldots$$

$$x_2 := x_1 + x_2 + x_3 + 2x_4 + 3x_5 + \ldots$$

Next, we require only two elements (first and second) to compute the mapping.

$$x_3 := F_1x_1 + F_2x_2$$

$$x_4 := x_3 + (F_2 - F_1)\, x_1 + F_1x_2$$

$$x_5 := x_3 + x_4$$

$$x_6 := x_4 + x_5$$

$$\cdots \quad \cdots \quad \cdots \quad \cdots$$

$$x_n := x_{n-2} + x_{n-1}$$

Finally, we construct two mappings as follows.

$$x_1 := x_3 - (F_2 - F_1)\, x_1 + F_1x_2$$

$$x_2 := x_3 - x_1$$

This sequence of assignments compute mapping E effectively, and the number of these assignments are $n+2$. □

Lemma 5. *Let $E : \mathbb{Z}^2 \longrightarrow \mathbb{Z}^2$ be a linear mapping defined as $E(x, y) \longrightarrow (mx+ny, px+qy)$. Let E can be computed by a sequence of linear assignments*

$$\begin{aligned} x &:= a_1x + b_1y \\ y &:= b_2x + a_2y \\ x &:= a_3x + b_3y \\ y &:= b_4x + a_4y \\ \vdots &:= \quad \vdots \quad \vdots \\ x &:= a_{k-1}x + b_{k-1}y \\ y &:= b_kx + a_ky \end{aligned}$$

where a_i, $b_i \in \mathbb{Z}, \forall\ 1 \leq i \leq k$, $k \in \mathbb{Z}$, and $k \geq 2$. Then for m, n, p, $q \in \mathbb{Z}$

$$m * q - p * n = a_1 * a_2 * a_3 \ldots a_{k-1} * a_k$$

where $m * q - p * n$ is the determinant of the matrix "A", that represents coefficients of mapping E.

$$A = \begin{pmatrix} m & n \\ p & q \end{pmatrix}$$

Proof. Suppose that Δ denotes the determinant of the matrix A, then we have to prove that

$$\Delta = a_1 * a_2 * a_3 \ldots a_{k-1} * a_k$$

Consider, the evaluation of the given sequence of linear assignments

$$x := a_1x + b_1y \tag{3.5}$$
$$y := b_2a_1x + (b_2b_1 + a_2)y \tag{3.6}$$

$$x := [a_3a_1 + b_3(b_2a_1)]x + [a_3b_1 + b_3(b_2b_1 + a_2)] \tag{3.7}$$
$$\begin{aligned} y := {} & [a_4(b_2a_1) + b_4\{a_3a_1 + b_3(b_2a_1)\}]x \\ & + [a_4(b_2b_1 + a_2) + b_4\{a_3b_1 + b_3(b_2b_1 + a_2))\}]y \end{aligned} \tag{3.8}$$

$$\begin{aligned} x := {} & [a_5\{a_3a_1 + b_3(b_2a_1)\} + b_5\{a_4(b_2a_1) + b_4(a_3a_1 + b_3(b_2a_1))\}]x \\ & + [a_5\{a_3b_1 + b_3(b_2b_1 + a_2)\} + b_5\{a_4(b_2b_1 + a_2) \\ & + b_4(a_3b_1 + b_3(b_2b_1 + a_2))\}]y \end{aligned} \tag{3.9}$$
$$\begin{aligned} y := {} & [a_6\{a_4(b_2a_1) + b_4(a_3a_1 + b_3(b_2a_1)\} + b_6\{a_5(a_3a_1 \\ & + b_3(b_2a_1) + b_5(a_4(b_2a_1) + b_4((a_3a_1 + b_3(b_2a_1)))\}]x \\ & + [a_6\{a_4(b_2b_1 + a_2) + b_4(a_3b_1 + b_3(b_2b_1 + a_2))\} \\ & + b_6\{a_5(a_3b_1 + b_3(b_2b_1 + a_2)) + b_5\{a_4(b_2b_1 + a_2) \\ & + b_4(a_3b_1 + b_3(b_2b_1 + a_2))\}\}]y \end{aligned} \tag{3.10}$$

$$\vdots := \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$$

$$\begin{aligned} x := [&a_{k-1}(\text{x-component of (k-3)th assignment}) \\ &+ b_{k-1}(\text{x-component of (k-2)th assignment})]x \\ &+ a_{k-1}(\text{y-component of (k-3)th assignment}) \\ &+ b_{k-1}(\text{y-component of (k-2)th assignment})]y \end{aligned} \tag{3.11}$$

$$\begin{aligned} y := [&a_k(\text{x-component of (k-2)th assignment}) \\ &+ b_k(\text{x-component of (k-1)th assignment})]x \\ &+ a_k(\text{y-component of (k-2)th assignment}) \\ &+ b_k(\text{y-component of (k-1)th assignment})]y \end{aligned} \tag{3.12}$$

Now,

$$\begin{aligned} \text{Determinant of the Matrix } &= [(\text{x-component of (k-1)th assignment}) \\ &* (\text{y-component of (k)th assignment})] \\ &- [(\text{y-component of (k-1)th assignment}) \\ &* (\text{x-component of (k)th assignment})] \\ &= m * q - p * n = \Delta \end{aligned} \tag{3.13}$$

Therefore, we can write

$$\begin{aligned} m = [&a_{k-1}(\text{x-component of (k-3)th assignment}) \\ &+ b_{k-1}(\text{x-component of (k-2)th assignment})]x \\ n = &a_{k-1}(\text{y-component of (k-3)th assignment}) \\ &+ b_{k-1}(\text{y-component of (k-2)th assignment})]y \\ p = [&a_k(\text{x-component of (k-2)th assignment}) \\ &+ b_k(\text{x-component of (k-1)th assignment})] = f + c \\ q = &a_k(\text{y-component of (k-2)th assignment}) \\ &+ b_k(\text{y-component of (k-1)th assignment})] = e + d \\ \text{where } f = &a_k(\text{x-component of (k-2)th assignment}) \\ c = &b_k(\text{x-component of (k-1)th assignment}) \\ e = &a_k(\text{y-component of (k-2)th assignment}) \\ d = &b_k(\text{y-component of (k-1)th assignment}) \end{aligned}$$

$$\Delta = m * q - n * p = me - nf + md - nc$$

Since $md - nc = 0$, Therefore $\Delta = me - nc$, that is

$$\begin{aligned}\Delta = a_{k-1}a_k[&\text{(x-component of (k-3)th assignment)}\\ &* \text{(y-component of (k-2)th assignment)}]\\ + a_k b_{k-1}[&\text{(x-component of (k-2)th assignment)}\\ &* \text{(y-component of (k-2)th assignment)}]\\ - a_{k-1}a_k[&\text{(x-component of (k-2)th assignment)}\\ &* \text{(y-component of (k-3)th assignment)}]\\ - a_k b_{k-1}[&\text{(y-component of (k-2)th assignment)}\\ &* \text{(x-component of (k-2)th assignment)}]\end{aligned}$$

After simplification, we will have

$$\begin{aligned}\Delta = a_{k-1}a_k[&\text{(x-component of (k-3)th assignment)}\\ &* \text{(y-component of (k-2)th assignment)}]\\ &- [\text{(x-component of (k-2)th assignment)}\\ &* \text{(y-component of (k-3)th assignment)}]\end{aligned} \tag{3.14}$$

Observe that it is again in the form of (a). Hence by continuing this process of evaluation, we will have

$$\Delta = a_k * a_{k-1} * a_{k-2} \ldots a_4 * a_3 * a_2 * a_1$$

□

Example 16. *Let $E : \mathbb{Z}^2 \longrightarrow \mathbb{Z}^2$ be a linear mapping defined as $E(x, y) \longrightarrow (33x + 307y, 103x + 610y)$. The coefficients of mapping can be represented as a matrix A of size 2×2 as follows.*

$$A = \begin{pmatrix} 33 & 307 \\ 103 & 610 \end{pmatrix}$$

$$\textit{Determinant } (A) = -11491$$

The mapping E can be computed by the following sequence of linear assignments

$$\begin{aligned} x &:= x + 9y \\ y &:= 3x + y \\ x &:= x + 3y \\ y &:= 2x + y \\ x &:= x + y \\ y &:= 8012x - 11491y \end{aligned}$$

$$\textit{Product of coefficients } a_i s = -11491$$

Chapter 4

In Situ Computation of Boolean Mappings

In this chapter, we describe the in situ computation of boolean mappings. A Boolean mapping describes how to determine a Boolean value output based on some logical combination from Boolean inputs. Boolean mappings are very important in the theory of complexity as well as in the design of circuits and chips for digital computers. The properties of Boolean mappings play a crucial role in cryptography, particularly in the design of symmetric key algorithms (a class of algorithms for cryptography that use boolean function keys for both decryption and encryption), *e.g.* Two fish, Serpent, Blowfish, CAST5, RC4, TDES, and IDEA.

Boolean mappings can be represented in propositional logic, or as multivariate polynomials over $GF(2)$. To perform numerical computations on logical symbols, was one of the work of George Boole in the theory of logic.

Boolean mappings have a number of applications in different areas including artificial intelligence, propositional logic, circuit design, electrical engineering, game theory, reliability theory and combinatorics. The representation of boolean functions as a boolean polynomial is an extensive method in boolean algebra. Boolean polynomials play an important role in Reed-Muller codes (Error Correcting codes). The widespread use in electronics of integrated circuits that include "modulo 2 adders" focuses the attention on the representation of Boolean functions in the form of polynomials. Every boolean function can be represented as a polynomial. Polynomial methods have been employed extensively in circuit complexity. Boolean polynomials occur either directly or as a tool in the problem of decomposing a boolean function. The decomposition of boolean function is considered to be an important problem in the design of logic circuits. The decomposition of switching function is very important. A switching function is $f : \{0,1\}^n \longrightarrow \{0,1\}$. Boolean polynomial have a large number of applications in various fields including graph theory, law, medicine, operations

research and spectroscopy[19, 39, 37, 3, 46].
Burckel et al. [11] have proved that any linear boolean mapping with input variables n can be computed with a double sequence of linear assignments of the same number of variables as given in the input. This result leads to a decomposition of boolean matrices and directed graphs. Burckel et al. also proved that every boolean mapping is decomposable in $4n - 3$ boolean functions.
Next, we start by explaining the basic concepts. Then, we explain the in situ computation of bijective boolean mappings. Then, on the basis of this result, we explain the in situ computation of general boolean mappings and verified the results. We quoted definitions that are necessary to prove these results. At the end, we introduce a new method for the case of boolean bijective mappings via algebraic operations over polynomials in GF2.

Boolean Mappings:

We describe here the basic concept of boolean mapping and we explain it with the help of an example and truth table.

Definition 4. *A Boolean function/mapping is a function/mapping f from the Cartesian product $\{0,1\}^n$ to $\{0,1\}$. Alternatively, we write $f : \{0,1\}^n \longrightarrow \{0,1\}$. The set $\{0,1\}^n$, by definition, the set of all n-tuples $(x_1, \cdots, x_n)$ where each x_i is either 0 or 1, is called the domain of f . The set $\{0,1\}$ is called the Co domain (or, sometimes, range) of f . There are 2^{2^n}, n-ary functions for every n.*

One way to represent a boolean function whose domain is finite uses a table. Each element x of the domain has a row of the table listing the domain element x and the corresponding function value $f(x)$. These tables make easy to understand and construct these functions from tables.

Example 17. *Consider the boolean mappings y_1, y_2, and y_3 from $\{0,1\}^3$ to $\{0,1\}$ defined as follows:*

$$y_1 = x_1 + x_2 * x_3$$
$$y_2 = x_1 + x_2 + x_1 * x_3 + x_1 * x_2 + x_2 * x_3$$
$$y_3 = x_3 + x_1 * x_2 + x_2 * x_3$$

The table (4.5) represents the functions y_1, y_2 and y_3.

x_1	x_2	x_3	y_1	y_2	y_3
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	1	1	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	0	1	1

Table 4.1: Truth Table 1

4.1 Computing Bijective Boolean Mappings

If one can compute bijective boolean mapping with an in situ program that consist of a set of assignments then the inverse bijective mapping can also be computed with the same number of assignments that can be obtained by inverting the assignments and rewriting from bottom to top.

Definition 5. *An in situ program that can compute a boolean mapping* $E : \{0,1\}^n \longrightarrow \{0,1\}^n$*, consists in a sequence of assignments of one bit component that can be written as*

$$x_j := f_j\left(x_1, \cdots, x_n\right)$$

where $f_j : \{0,1\}^n \longrightarrow \{0,1\}$ *is a linear mapping and* j *is the index for the input variables.*

Theorem 1. *A bijective mapping* E *defined over* $\{0,1\}^n$ *can be computed by an in situ program of the form.*

$$f_n, f_{n-1}, \cdots, f_3, f_2, f_1, g_2, g_3, \cdots, g_{n-1}, g_n$$

and of length $2n-1$

Proof. We proceed by induction over n.
For $n = 1$, the program will be $x_1 := f_1(x_1)$.
Now, we have to prove that the statement is true for $n > 1$. We will make use of bipartite graph. Suppose that $G = (X, Y, A)$ be the bipartite multi-edges graph defined by $X = Y = \{0,1\}^{n-1}$, and $(x, y) \in X \times Y$ is in A with the label $x_n y_n$, for $x_n y_n \in \{0,1\}$ if and only if $E(x, x_n) = (y, y_n)$.
The degree of vertices of graph G will be exactly 2 due to the bijection E and graph G will be the union of disjoint even cycles due to the reason that it is 2-color-able regular bipartite graph. Therefore, by definition graph G is 2-color-able[25]. We mention that this is a particular case of a general result by König on regular graphs [34], from which this proof can be generalized to any mapping on a finite set. Let us colour the edges of G with

elements of $\{0,1\}$. Now define two mappings E^0, E^1 on $\{0,1\}^{n-1}$ and two mappings f_n, g_n from $\{0,1\}^n$ to $\{0,1\}$ as follows. For each colour $i \in \{0,1\}$ and every edge (x,y) with colour i and labeled $x_n y_n$, define

$$E^i(x) = y$$
$$f_n(x, x_n) = i$$
$$g_n(y, i) = y_n$$

By construction, any mapping E^i is bijective on $\{0,1\}^{n-1}$. Then under induction hypothesis, each E^i admits an in situ program of the form:

$$f^i_{n-1}, \cdots, f^i_2, f^i_1, g^i_2, \cdots, g^i_{n-1}$$

Define for every $i \in \{0,1\}$ and $x \in \{0,1\}^{n-1}$.

$$f_j(x,i) = f^i_j(x)\ for j = n-1, \cdots, 1$$
$$g_j(x,i) = g^i_j(x)\ for j = 2, \cdots, n-1$$

In other words,

$$f_j(x, x_n) = x_n.f^1_j(x) + (1 + x_n).f^0_j(x)\ for j = n-1, \cdots, 1$$
$$g_j(x, x_n) = x_n.g^1_j(x) + (1 + x_n).g^0_j(x)\ for j = n-1, \cdots, 1$$

By construction we have obtained an in situ program of length $2(n-1) - 1 + 2 = 2n - 1$ for E:

$$\begin{aligned}
x_n &:= f_n(x_1, \cdots, x_n) \\
x_{n-1} &:= f_{n-1}(x_1, \cdots, x_n) = f^i_{n-1}(x_1, \cdots, x_{n-1}) \\
x_{n-2} &:= f_{n-2}(x_1, \cdots, x_n) = f^i_{n-2}(x_1, \cdots, x_{n-1}) \\
&\vdots \qquad\qquad \vdots \qquad\qquad \vdots \qquad\qquad \vdots \\
x_2 &:= f_2(x_1, \cdots, x_n) = f^i_2(x_1, \cdots, x_{n-1}) \\
x_1 &:= f_1(x_1, \cdots, x_n) = f^i_1(x_1, \cdots, x_{n-1}) \\
x_2 &:= g_2(x_1, \cdots, x_n) = g^i_2(x_1, \cdots, x_{n-1}) \\
&\vdots \qquad\qquad \vdots \qquad\qquad \vdots \qquad\qquad \vdots \\
x_{n-2} &:= g_{n-2}(x_1, \cdots, x_n) = g^i_{n-2}(x_1, \cdots, x_{n-1}) \\
x_{n-1} &:= g_{n-1}(x_1, \cdots, x_n) = g^i_{n-1}(x_1, \cdots, x_{n-1}) \\
x_n &:= g_n(x_1, \cdots, x_n) = g_n(y_1, \cdots, y_{n-1}, i)
\end{aligned}$$

In other words at the first step, the component x_n equals a colour i, Then $E^i(x_1, \cdots, x_{n-1})$ is computed by induction in 2(n-1)-1 steps. At the step before last, we have $(x_1, \cdots, x_{n-1}) = (y_1, \cdots, y_{n-1})$. At the last step we have $x_n = y_n$. □

We describe below, two examples to explain the theorem 1.

Example 18. *Consider the bijective boolean mapping E_1 defined over $\{0, 1\}^3$ as given below.*

x_1	x_2	x_3
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

$\xrightarrow{E_1}$

x_1	x_2	x_3
0	0	0
1	1	1
0	0	1
0	1	1
1	0	1
0	1	0
1	0	0
1	1	0

Table 4.2: Table

The bijective mapping E_1 can be computed by performing the following operations.

$$x_1 := 1 + x_2 + x_1x_3 \qquad (f_1')$$
$$x_2 := 1 + x_1 + x_2 + x_1x_3 + x_2x_3 + x_1x_2 \qquad (f_2')$$
$$x_3 := x_1 + x_2 + x_3 + x_2x_3 \qquad (f_3')$$
$$x_2 := x_1 + x_2 + x_1x_3 \qquad (g_2')$$
$$x_1 := 1 + x_1 + x_2 + x_3 + x_2x_3 \qquad (g_1')$$

After performing the first three operations f_1', f_2', f_3' , the mapping E_1 transform into the boolean mapping that is still bijective then after performing operations g_2', g_1' we get the required mapping.

x_1	x_2	x_3
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

$\xrightarrow{f_1',\, f_2',\, f_3'}$

x_1	x_2	x_3
1	1	0
1	1	1
0	0	1
0	1	1
1	0	1
0	1	0
0	0	0
1	0	0

$\xrightarrow{g_2',\, g_1'}$

x_1	x_2	x_3
0	0	0
1	1	1
0	0	1
0	1	1
1	0	1
0	1	0
1	0	0
1	1	0

Table 4.3: Table

We describe another example to explain all steps in the computation of boolean bijective mapping.

Example 19. *Consider the bijective boolean mapping E_2 defined over $\{0, 1\}^3$ as given below.*

x_1	x_2	x_3
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

$\xrightarrow{E_2}$

x_1	x_2	x_3
1	1	1
0	1	1
1	1	0
0	0	1
0	0	0
1	0	1
1	0	0
0	1	0

Table 4.4: Table

The given bijective mapping E_2 can be computed by performing the following operations.

$$x_1 := 1 + x_1 + x_2 + x_3 + x_2x_3 \quad (f_1')$$
$$x_2 := x_1 + x_2 + x_1x_3 \quad (f_2')$$
$$x_3 := x_3 + x_1x_2 \quad (f_3')$$
$$x_2 := x_2 + x_3 + x_1x_3 \quad (g_2')$$
$$x_1 := x_1 + x_2 + x_2x_3 \quad (g_1')$$

We illustrate step by step the computation of bijective mapping E_2 as given below.

Lemma 6. *Every assignment $x_i := f_i(x_1, \cdots, x_n)$ performed in an in situ program to compute a bijective mapping must be linear in x_i,i.e.*

$$f_i(x_1, \cdots, x_n) = x_i + h(x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_n)$$

Proof. There are two possible cases for the mapping $f_i\,(x_1, x_2, x_3 \cdots, x_n)$ for $x_1 \in \{0, 1\}$ *i.e.* $f_i\,(0, x_2, x_3 \cdots, x_n)$ or $f_i\,(1, x_2, x_3 \cdots, x_n)$
For each of these cases, there exist two possibilities for the output value *i.e.*

$$f_i\,(0, x_2, x_3 \cdots, x_n) = 0 \quad \text{(a)}$$
$$f_i\,(0, x_2, x_3 \cdots, x_n) = 1 \quad \text{(b)}$$
$$f_i\,(1, x_2, x_3 \cdots, x_n) = 0 \quad \text{(c)}$$
$$f_i\,(1, x_2, x_3 \cdots, x_n) = 1 \quad \text{(d)}$$

In the computation of bijection, exactly, one of the equations from (a,b) and one of the equations from (c,d) will hold.
Since the pair of equations (a,c) and (b,d) cannot hold simultaneously due to the bijection,

x_1	x_2	x_3
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

$\xrightarrow{f_1'}$

x_1	x_2	x_3
1	0	0
0	0	1
0	1	0
0	1	1
0	0	0
1	0	1
1	0	0
1	1	0

$\xrightarrow{f_2'}$

x_1	x_2	x_3
1	1	0
0	0	1
0	1	0
0	1	1
0	0	0
1	0	1
1	0	0
1	1	0

$\xrightarrow{f_3'}$

x_1	x_2	x_3
1	1	1
0	0	1
0	1	0
0	1	1
0	0	0
1	0	1
1	0	0
1	1	0

$\xrightarrow{g_2'}$

x_1	x_2	x_3
1	1	1
0	1	1
0	1	0
0	0	1
0	0	0
1	0	1
1	0	0
1	1	0

$\xrightarrow{g_1'}$

x_1	x_2	x_3
1	1	1
0	1	1
1	1	0
0	0	1
0	0	0
1	0	1
1	0	0
0	1	0

Table 4.5: Table

therefore, the only two possible pairs of equations (a,d) and (b,c) will hold. But the pair (a,d) will define the mapping

$$f_i\left(x_1, x_2, x_3 \cdots, x_n\right) = x_1$$

and the pair (b,c) defines

$$f_i\left(x_1, x_2, x_3 \cdots, x_n\right) = x_1 + 1$$

Therefore for all cases

$$f_i(x_1, \cdots, x_n) = x_1 + h(x_2, \cdots, x_n)$$

$$\text{where, } h(x_2, \cdots, x_n) = 0, \text{ for the pair (a,d)}$$

$$\text{and, } h\left(x_2, \cdots, x_n\right) = 1, \text{ for the pair (b,c)}$$

. □

4.2 Computing General Boolean Mappings

In this section, we describe the in situ computation of mappings over $\{0,1\}^n$ with the supposition that two different vectors may have the same image that is not the case for

bijective mappings. we will describe upper bound over the length of the program that we use to compute these mappings. We describe some basic definitions that will help to prove the assertions. We will use the mapping $\phi : \{0,1\}^n \longrightarrow \{2^n\}$ defined as

$$\phi(x_1, x_2, x_3, \ldots, x_n) = 2^{n-1}.x_n + \ldots + 2^2.x_3 + 2.x_2 + x_1, \forall n > 0$$

and we also make use of simple mapping (that can be computed with n assignments) to prove the upper bound over the length of program.
The idea consist in decomposing the mapping E of $\{0,1\}^n$ in $F \circ P \circ G$ where F and G are bijective and P is simple. This will lead to a sequence of $5n-4$ assignments to compute the mapping E.

Definition 6. *For every $M > 0$, a mapping f on $\{0, \ldots, M\}$ is a step mapping if for every $0 \leq x \leq y \leq M$:*

$$0 \leq f(y) - f(x) \leq y - x$$

In particular one has $f(x) \leq f(x+1) \leq f(x) + 1$. A mapping E on $\{0,1\}^n$ is simple if the mapping E' on $[2^n]$ such that

$$E(x) = y \Leftrightarrow E'(\phi(x)) = \phi(y)$$

is a step mapping.

Proposition 5. *Every simple mapping E on $\{0,1\}^n$, $\forall\, n > 0$, is computed by an in situ program of length n of the form*

$$p_1, p_2, \ldots, p_n$$

precisely, for $E(x_1, x_2, x_3, \ldots, x_n) = y_1, y_2, y_3, \ldots, y_n$ and for each $i = 1, 2, \ldots n$

$$p_i(y_1, \ldots, y_{i-1}, x_i, \ldots, x_n) = y_i$$

Proof. For n input variables the n assignments are minimal. Therefore each function p_i must return its correct final value to each of its corresponding component x_i. This method for in situ computation of simple mapping is unique possible. However the correctness of this method is still remains to prove due to the fact that the mappings being computed are simple. □

Next, we describe a definition that will use to decompose a mapping as a composition of three mappings.

Definition 7. *For $n > 0$, let E be a mapping on $\{0,1\}^n$. A decomposition of E is a triple of mapping (F, P, G) on $\{0, 1\}^n$, such that $E = F \circ P \circ G$ with F, G bijective and P simple.*

For a given mapping E on $\{0,1\}^n$, the decomposition can be build in a way that we can group the vectors with same images via a bijective mapping G which give intermediary consecutive images in lexicographical order to vectors with same final images. This will actually maps the sets $E^{-1}(x)$ for $x \in \{0,1\}^n$ onto consecutive intervals of $[2^n]$ via $\phi \circ G$.

The second step consists in the identification of vectors with the same final image via a simple mapping P such that $\phi \circ P \circ \phi^{-1}$ maps consecutive intervals of $[2^n]$ on consecutive values of $[2^n]$. The third step consists in the attribution of the correct final image value to each vector obtained by $P \circ G$ via another bijective mapping F, that is $F(P(G(x))) = E(x)$. Then F is completed arbitrarily to a bijective mapping on $\{0,1\}^n$.

Corollary 1. *Every mapping E on $\{0,1\}^n$ is computed by an in situ program of length $5n-4$ of the form*

$$f_1, \ldots, f_n, g_{n-1}, \ldots, g_1, p_2, \ldots, p_{n-1}, f'_n, \ldots, f'_1, g'_2, \ldots, g'_n$$

Proof. The proof can be constructed easily by using the above procedure of decomposition. Suppose that $E = F \circ P \circ G$, *i.e.* the triple (F, P, G) be the decomposition of E, where F, G are bijective mappings and P is a simple mapping. It has been proved in theorem 1 that both F and G can be computed by a sequence of $2n-1$ assignments respectively.
Similarly, the simple mapping P can be computed by a sequence of n assignments as proved in proposition 5. Combining these number of assignments E, then can be computed by a sequence of $5n-2$ assignments, of the form

$$f_1, \ldots, f_n,\ g_{n-1}, \ldots, g_1,\ p_2, \ldots, p_{n-1}, f'_n, \ldots, f'_1, g'_2, \ldots, g'_n$$

Two more assignments can be reduced by selecting a sequence of $2n-1$ assignments in such a way that it begins with the first variable for G and with the last for F. Moreover, two successive assignments of the same component can be combined into a single assignment. Therefore, g_1, p_1 can be replaced by a single assignment g_1 and p_n, f'_n can be replaced by the assignment f'_n.
Finally, we will get a sequence of $5n-4$ assignments. □

Definition 8. *A mapping $v : [2^n] \longrightarrow \mathbb{N}$ defined by $v(0) + \ldots + v(2^n - 1) = 2^n$ is called valuation. We denote $v(l)$, the value of $l \in [2^n]$, and by extension $v(A) = \sum_{l \in A} v(l)$ for $A \subseteq [2^n]$.*
A valuation v is called boolean compatible if $\forall\, i$ such that $0 \leq i \leq n$ and all $0 \leq j < 2^{n-i}$.

$$\sum_{j2^i \leq l < (j+1)2^i} v(l) = 0 \ mod\ 2^i$$

Definition 9. *Let v be the valuation, a mapping $P_v : \{0,1\}^n \longrightarrow \{0,1\}^n$ is called projection of v such that for l from 0 to $2^n - 1$, $v(l)$ consecutive elements of $\{0,1\}^n$ are mapped onto $\phi^{-1}(l)$, beginning with $(0, \ldots, 0) \in \{0,1\}^n$ for the first l with $v(l) \neq 0$. P_v is a simple mapping.*

Lemma 7. *Let v be a boolean-compatible valuation.*
For $I_{i,j} = \phi^{-1}([j2^i, (j+1)2^i - 1])$ and for some $k, k' \in [2^{n-i}]$,
where $0 \leq i \leq n$ and $j \in [2^{n-i}]$, we have

$$P_v^{-1}(I_{i,j}) = \bigcup_{k \leq l \leq k'} I_{i,l}$$

Proof. Suppose that the converse image of an interval of $[2^n]$ by the mapping ϕ is an interval of $\{0,\,1\}^n$. Also, the converse image of an interval $\{0,\,1\}^n$ by the mapping P_v is an interval of $\{0,\,1\}^n$ due to the fact that P_v is a simple mapping. we have $|P_v^{-1}(I_{i,j})| = \sum_{j2^i \le l < (j+1)2^i} v(l)$ by definition of P_v
For all $l \in \phi(I,i,j)$, if $v(l) = 0$ then $P_v^{-1}(I_{i,j})$ may be empty. we have $\sum_{j2^i \le l < (j+1)2^i} v(l) = 0 \ mod\ 2^i$, due to the fact that v is boolean compatible. Thus $|P_v^{-1}(I_{i,j})| = 0 \ mod\ 2^i$. By induction on j for a fixed i, the result can be proved. If $j = 0$ then $|P_v^{-1}(I_{i,0})| = k.2^i$ for some $k \in [2^{n-i}]$. If $P_v^{-1}(I_{i,j})$ is not empty, then it is an interval of $\{0,\,1\}^n$ containing $(0,\ldots,0)$ by definition of P_v. Since this interval has a lenght $k.2^i$ multiple of 2^i, it is of the form $\bigcup_{0\le l\le k} I_{i,l}$. If the property is true for all l with $0 \le l < j$, then $P_v^{-1}(\bigcup_{0\le l<j} I_{i,l}) = \bigcup_{0\le l<j'} I_{i,l}$. Since $|P_v^{-1}I_{i,j}| = k.2^i$ for some $k \in [2^{n-i}]$, we must have $P_v^{-1}(\bigcup_{0\le l\le j} I_{i,l}) = \bigcup_{0\le l\le j'+k} I_{i,l}$, hence $P_v^{-1}(I_{i,j}) = \bigcup_{j'<l\le j'+k'} I_{i,l}$

□

Lemma 8. *Let v be a boolean compatible valuation. Let $a, b \in \{0,1\}^n$, and let $1 \le i \le n.$, If for all $l \ge i$ we have $a_l = b_l$, then for all $l \ge i$ we have $P_v(a)_l = P_v(b)_l$*

Proof. We want to prove that $a_l = b_l \Longrightarrow P_v(a)_l = P_v(b)_l$.
Suppose that $P_v(a)_l \neq P_v(b)_l$. Using lemma 7 there exist l, where, $k \le l \le k'$ such that $a_l \neq b_l$.
A contradiction. Hence our supposition is wrong. □

Proposition 6. *Let E be a bijective mapping on $\{0,1\}^n$ computable by an in situ program $f_1,\ldots,f_n$. Then the mapping $E \circ P_v$ is computed by an in situ program of the form $p_1, p_2, \ldots, p_n$, where v is a boolean compatible valuation. Precisely, for $E(x_1, x_2, \ldots, x_n) = (y_1, \ldots, y_n)$ and for each $i = 1, 2, \ldots, n$: $p_i(y_1, \ldots, y_{i-1}, x_i, \ldots, x_n) = y_i$*

Proof. To prove the assertion, we use the same argument as used for the computation of simple mappings *i.e.* due to the fact that the number of assignments is minimal, necessarily each function p_i must give its correct final value to each component x_i. But the correctness of the method is still to be proved.
On contrary basis, suppose that, at some step i, two different vectors x, x' become the same vector, say z, whereas their final images $y = E \circ P_v(x)$ and $y' = E \circ P_v(x')$ are different. Because, now, they become the same vector, one has $y_{i-1} = y'_{i-1} = z_{i-1}, \ldots, y_1 = y'_1 = z_1$. On the other side, one has $x_n = x'_n = z_n, \ldots, x_i = x'_i = z_i$ which implies, by lemma 8 $P_v(x)_l = P_v(x')_l = z'_l$ for all $l \ge i$.
Assume $P_v(x) \neq P_v(x')$. Since E is bijective, then the mapping E' programmed by the sequence $f_1, \ldots, f_{i-1}$ is also bijective. But since E is programmed by $f_1, \ldots, f_n$, where

each component is modified at most once, we must have $E'(P_v(x)) = E'(P_v(x')) = (z'_n, \ldots, z'_i, z'_{i-1}, \ldots, z_1)$. A contradiction. Hence $P_v(x) = P_v(x')$ and $y = y'$ □

Definition 10. *A boolean compatible decomposition of E defined over* $\{0,1\}^n$ *for* $n > 0$ *is a triple of mappings* (F, P, G) *on* $\{0,1\}^n$*, such that* $E = F \circ P \circ G$ *with F, G bijective and P is the projection* P_v *of a boolean compatible valuation* $v = v' \circ \sigma$*, where* $v'(l)$ *is the number of elements of which image by E is* $\phi(l)$*.*

We describe an algorithm in the following that build one of the mapping as described in the above definition.

Algorithm:

Define a valuation v by $v(l) = |E^{-1}(\phi(l))|$
Define a permutation σ such that $v \circ \sigma$ is boolean-compatible
Define a bijection G of $\{0,1\}^n$ compatible with $v \circ \sigma$
Set i=0;
for *l from 0 to* $2^n - 1$ **do**
  for *j from 1 to* $v \circ \sigma(l)$ **do**
    for *every* $x \in \{0,1\}^n$ *with* $E(x) = \phi(l)$ **do**
      G(x):=$\phi^{-1}(i)$;
      i:=i+1;
    end
  end
end
Define $P = P_v$;
Define a bijection F of $\{0,1\}^n$ such that $E = F \circ P \circ G$;
for *Every* $x \in \{0,1\}^n$ **do**
  F(P(G(x))):=E(x);
end
Complete the definition of F, keeping F bijective.

Theorem 2. *Every mapping E on* $\{0,1\}^n$ *is computed by an in situ program of length* $4n-3$ *of the form*

$$f''_1, \ldots, f''_n, g''_{n-1}, \ldots, g''_2, f_1, \ldots, f_n, g'_{n-1}, \ldots, g'_1$$

Proof. Let (F, P, G) be a boolean-compatible decomposition of E (definition 10) . Let $f'_1, \ldots, f'_n, g'_{n-1}, \ldots, g'_1$, resp $f''_1, \ldots, f''_n, g''_{n-1}, \ldots, g''_1$ be the sequence of $2n-1$ assignments computing F, resp, G, by theorem 1. Let F' be the mapping on $\{0,1\}^n$ defined by the sequence $f'_1, \ldots, f'_n$. Then by proposition 6 the mapping $F' \circ P$ is computed by a sequence of n assignments $f_1, \ldots, f_n$. Then E is computed by the sequence of $4n-2$ assignments $f''_1, \ldots, f''_n, g''_{n-1}, \ldots, g''_1, f_1, \ldots, f_n, g'_{n-1}, \ldots, g'_1$, where the indices are the indices of the concerned variables. Of course g''_1, f_1 can be replaced by a single g_1. Then we have a sequence of $4n-3$ assignments. □

Next, we introduce a method for the case of boolean bijective mappings via algebraic operations over polynomials in $GF(2)$.

4.3 A First Tool

In this section, we describe a method that computes a boolean function linear in x_1 and y_1, where y_1, y_2, y_3, ..., y_n are bijective mappings over x_1, x_2, x_3, ..., x_n and x_1, x_2, x_3, ..., x_n are inverse bijective mappings over y_1, y_2, y_3, ..., y_n. In the context of Lemma 6, this will lead to the existence of in situ computation of bijective boolean mappings. Further investigation will lead to design an efficient algorithm that will help to reduce the complexity and ultimately to reduce the memory usage.

4.3.1 The Algorithm

Consider boolean bijective mappings as given below:

$$\begin{aligned}
y_1 &:= f_1(x_1, x_2, x_3, \cdots, x_n)\\
y_2 &:= f_2(x_1, x_2, x_3, \cdots, x_n)\\
y_3 &:= f_3(x_1, x_2, x_3, \cdots, x_n)\\
&\cdots \quad \cdots \quad \cdots \quad \cdots\\
y_{n-1} &:= f_{n-1}(x_1, x_2, x_3, \cdots, x_n)\\
y_n &:= f_n(x_1, x_2, x_3, \cdots, x_n)
\end{aligned}$$

We are interested to compute a polynomial that is linear both in x_1 and y_1. We can compute such kinds of polynomial by proceeding the following steps.
Step-1:
Compute the inverse mappings of each of y_1, y_2, y_3, $\cdots, y_{n-1}$, y_n. These inverse mappings can be written as

$$\begin{aligned}
x_1 &:= g_1(y_1, y_2, y_3, \cdots, y_n)\\
x_2 &:= g_2(y_1, y_2, y_3, \cdots, y_n)\\
x_3 &:= g_3(y_1, y_2, y_3, \cdots, y_n)\\
&\cdots \quad \cdots \quad \cdots \quad \cdots\\
x_{n-1} &:= g_{n-1}(y_1, y_2, y_3, \cdots, y_n)\\
x_n &:= g_n(y_1, y_2, y_3, \cdots, y_n)
\end{aligned}$$

Step-2:
Compute the possible products of permutations of $y_1,\ y_2,\ y_3,\ \cdots, y_{n-1},\ y_n$.

$$
\begin{array}{l}
y_1y_2, y_1y_3, \cdots, y_1y_n \\
y_2y_3, y_2y_4, \cdots, y_2y_n \\
y_3y_4, y_3y_5, \cdots, y_3y_n \\
\cdots \quad \cdots \quad \cdots \quad \cdots \\
y_1y_2y_3, y_1y_2y_4, \cdots, y_1y_2y_n \\
y_1y_3y_4, y_1y_3y_5, \cdots, y_1y_3y_n \\
\cdots \quad \cdots \quad \cdots \quad \cdots \\
y_1y_2y_3y_4, y_1y_2y_3y_5, \cdots, y_1y_2y_3y_n \\
y_1y_2y_3y_4y_5, y_1y_2y_3y_4y_6, \cdots, y_1y_2y_3y_4y_n \\
\cdots \quad \cdots \quad \cdots \quad \cdots \\
\cdots \quad \cdots \quad \cdots \quad \cdots
\end{array}
$$

Step-3: Compute the possible products of permutations of $x_1,\ x_2,\ x_3,\ \cdots, x_{n-1},\ x_n$ *e.g.*

$$
\begin{array}{l}
x_1x_2, x_1x_3, \cdots, x_1x_n \\
x_2x_3, x_2x_4, \cdots, x_2x_n \\
x_3x_4, x_3x_5, \cdots, x_3x_n \\
\cdots \quad \cdots \quad \cdots \quad \cdots \\
x_1x_2x_3, x_1x_2x_4, \cdots, x_1x_2x_n \\
x_1x_3x_4, x_1x_3x_5, \cdots, x_1x_3x_n \\
\cdots \quad \cdots \quad \cdots \quad \cdots \\
x_1x_2x_3x_4, x_1x_2x_3x_5, \cdots, x_1x_2x_3x_n \\
x_1x_2x_3x_4x_5, x_1x_2x_3x_4x_6, \cdots, x_1x_2x_3x_4x_n \\
\cdots \quad \cdots \quad \cdots \quad \cdots \\
\cdots \quad \cdots \quad \cdots \quad \cdots
\end{array}
$$

We want to obtain a polynomial of the form

$$y_1 + h_1\left(y_2,\ y_3,\ y_4, \cdots, y_n\right) = x_1 + h_2\left(x_2,\ x_3,\ x_4, \cdots, x_n\right) \tag{4.1}$$

We can introduce coefficients $(a_1,\ a_2,\ a_3, \cdots, a_i, \cdots,)$ to generate an expression of the form

$$y_1 + a_1y_2 + a_2y_3 + a_3y_2y_3 + \cdots \tag{4.2}$$

Eliminate the products involving x_1, and establish a system of equations in variables, $a_1, a_2, \cdots, a_i, , \cdots$. Substituting back these values of $a_1, a_2, \cdots, a_i, , \cdots$ in (4.2), we will have the required polynomial.

4.3.2 Explanation and Construction

Consider the boolean bijective mappings as given below:

$$y_1 := 1 + x_2 + x_1x_3$$
$$y_2 := 1 + x_1 + x_2 + x_1x_2 + x_2x_3 + x_1x_3$$
$$y_3 := x_1 + x_2 + x_3 + x_2x_3$$

These mappings can be represented in the form of a table as given below: To compute the

x_1	x_2	x_3	y_1	y_2	y_3
0	0	0	1	1	0
0	0	1	1	1	1
0	1	0	0	0	1
0	1	1	0	1	1
1	0	0	1	0	1
1	0	1	0	1	0
1	1	0	0	0	0
1	1	1	1	0	0

Table 4.6: Truth Table 2

polynomial that is linear in x_1 as well as in y_1, we will proceed as follows.
Compute the possible products as given below:

$$y_1y_2 := 1 + x_1 + x_2 + x_1x_2$$
$$y_1y_3 := x_1 + x_3 + x_1x_2 + x_2x_3$$
$$y_2y_3 := x_3 + x_1x_3$$
$$y_1y_2y_3 := x_3 + x_2x_3 + x_1x_3 + x_1x_2x_3$$

Compute the Inverse mappings:

$$x_1 := 1 + y_1y_3 + y_2y_1 + y_3$$
$$x_2 := 1 + y_1y_3 + y_2y_3 + y_2$$
$$x_3 := y_1 + y_2 + y_1y_3$$

Compute the possible products:

$$x_1x_2 := 1 + y_2 + y_3 + y_2y_3$$
$$x_1x_3 := y_1 + y_2 + y_1y_3 + y_2y_3$$
$$x_2x_3 := y_1 + y_2y_3 + y_1y_3 + y_2y_1$$
$$x_1x_2x_3 := y_1 + y_1y_3 + y_2y_1 + y_1y_3y_2$$

For α, β and γ we can make an expression

$$y_1 + \alpha y_2 + \beta y_3 + \gamma y_3 \tag{4.3}$$

$$\begin{aligned}
&= (1 + x_2 + x_1 x_3) \\
&+ \alpha\,(1 + x_1 + x_2 + x_1 x_2 + x_2 x_3 + x_1 x_3) \\
&+ \beta\,(x_1 + x_2 + x_3 + x_2 x_3) \\
&+ \gamma\,(x_3 + x_1 x_3)
\end{aligned}$$

We can establish a system of equations by comparing the coefficients of unwanted products *e.g.* $x_1 x_3$ etc.

$$\begin{aligned}
1 + \alpha + \gamma &= 0 \\
\alpha &= 0 \\
\alpha + \beta &= 1 \\
\implies \beta = 1, \gamma &= 1
\end{aligned}$$

Substituting the values of α, β and γ in (4.3), we get the required solution. Hence the solution that is linear in x_1 and y_1 is given as under.

$$x_2 x_3 + x_1 := 1 + y_1 + y_2 y_3 + y_3$$

Chapter 5

Maple Implementation

In this chapter, we describe complete code of algorithms that we have discussed in the previous chapters. We implemented these algorithms in Maple language.

5.1 Algorithm {In situ computation over $\mathbb{Z}$} :

- In this section, we describe an In Situ algorithm that can compute linear mappings sequentially over the set of integers $\mathbb{Z}$. We split the algorithm in three parts so that it is convenient to illustrate. This algorithm takes a square matrix as input and return the corresponding sequence of assignments as out put. Moreover it return the number of assignments used.

Input: Square matrix and its size
Output: Assignments
Assignments:=**proc()**;
Local variables i, j, s;
Global variables P, M, N;
for *i from N by* -1 *to* 1 **do**
  $s := 0$;
  for *j to N* **do**
    $s := s + M[i, j] * x[j]$;
  end
  $P := [op(P), x[i] = s]$;
end
End proc:

Algorithm 1: First part of the Algorithm

```
Input: Square matrix and its size
Output: Lower Triangular Matrix
Lower Triangular:=proc();
Local variables i, j, a;
Global variables P, M, N, x, X, Z;
P:=[];
This section will triangulate the matrix;
for i from 1 to N do
    for j from i+1 to N do
        Coloperations(i,j);
        print(i,j,M);
    end
end
print("Lower Triangular Matrix");
print(M);
print("Assignments are:");
Assignments();
print(P);
print("Number of assignments=",nops(P));
print("Verification");
for i to N do
    X[i] := Z[i];
end
for a in P do
    for i to N do
        if op(1, a) = x[i] then
            X[i] := eval(subs(x = X, op(2, a)));
        end
    end
end
for i to N do
    print(X[i]);
end
End proc;
```

Algorithm 2: Second part of the Algorithm

We split the algorithm in three parts so that it could be adjusted on the page. Next, we will describe the third part of the algorithm, that take Square matrix and its size as input, perform column operations and return the corresponding sequence of assignments.

Local variables a, b, c, cc, d, dd, k, **Global variables** P, M, N;
while $M[i,j] \neq 0$ **do**
  $a := M[i,i]$;
  $b := M[i,j]$;
  if $b < 0$ **then**
    $P := [op(P), x[j] = -x[j]]$;
    for k *to* N **do**
      $M[k,j] := -M[k,j]$;
    end
  end
  else if $a < 0$ **then**
    $P := [op(P), x[i] = -x[i]]$;
    for k *to* N **do**
      $M[k,i] := -M[k,i]$;
    end
  end
  else if $a > b$ **then**
    $cc := trunc(a/b)$;
    $dd := cc - 1$;
    $d := a \bmod b$;
    if $d = 0$ **then**
      $P := [op(P), x[j] = dd * x[i] + x[j]]$;
      for k *to* N **do**
        $M[k,i] := M[k,i] - dd * M[k,j]$;
      end
    end
    else
      $P := [op(P), x[j] = cc * x[i] + x[j]]$;
      for k *to* N **do**
        $M[k,i] := M[k,i] - cc * M[k,j]$;
      end
    end
  end
  else if $a = 0$ **then**
    $P := [op(P), x[j] = x[i] + x[j]]$;
    for k *to* N **do**
      $M[k,i] := M[k,i] - M[k,j]$;
    end
  end
  else if $b >= a$ **then**
    $c := trunc(b/a)$; $P := [op(P), x[i] = x[i] + c * x[j]]$;
    for k *to* N **do**
      $M[k,j] := M[k,j] - c * M[k,i]$;
    end
  end
end

5.2 Algorithm {Computing Boolean polynomial} :

- The Algorithm construct truth tables and then bijective boolean mappings from these truth tables. It forms a system of equations and finally return possible polynomials that are linear both in x_1 and y_1.

N:=3 (any integer); $M := 2^N$;
binary:= **proc(a,k)**;
local variables i,l,b;
b:=a; l:=[];
for *i to k* **do**
| l:=[b mod 2, op(l)];
| $b := trunc(b/2)$;
end
l;
End proc:

Algorithm 3: A part of the Algorithm 5.2

binaryL:= **proc(a,k)**;
local variables i,l,b;
b:=a; l:=[];
for *i to k* **do**
| l:=[op(l),b mod 2];
| b:=trunc(b/2);
end
l;
End proc: **for** *i to M* **do**
| X[i]:=binary(i-1,N);
end
End proc:

Algorithm 4: A part of the Algorithm 5.2

```
sous:=proc(f, p);
Local variables a, ok;
ok:=false; for a in f do
    if convert(expand(a + p), mod2) = 0 then
        ok:=true;
    end
end
ok;
End Proc:
```

Algorithm 5: A part of the Algorithm 5.2

```
fctx:= proc(l);
local variables i, j, k, s, p;
global variable X;
s := 0;
for i to M do
    if l[i] = 1 then
        p:=1;
        for k from 1 to N do
            if X[i][k]=1 then
                p := p * x[k] else
                p := p * (1 + x[k])
            end
            ;
        end
        s := s + p + p * s;
    end
end
s:=convert(expand(s), mod2);
s;
End proc:
```

Algorithm 6: A part of the Algorithm 5.2

```
fcty:=proc(l);
Local variables i, j, k, s, p;
Global variable Y;
s:=0;
for i to M do
    if l[i]=1 then
        p:=1;
        for k from 1 to N do
            if Y[i][k]=1 then
                p := p * y[k];
                else
                 |  p := p * (1 + y[k])
                end
                ;
            end
        end
        s := s + p + p * s;
    end
end
s := convert(expand(s), mod2);
s;
End proc:
```

Algorithm 7: A part of the Algorithm 5.2

```
test:=proc(f) ;
for i to M do
| Y[i]:=binary (f[i],N);
end
print("————————————————————————————————————————————————————————————");
print(f);
for i to M do
| print(X[i],Y[i]);
end
for j to N do
| l:=[];
| for i to M do
| | l:=[op(l),Y[i][j]];
| end
| YY[j]:=fctx(l);
end
for j to N do
| l:=[];
| for i to M do
| | l:=[op(l),X[i][j]];
| end
| XX[j]:=fcty(l);
end
print("MAPPING");
s:=[];
for j to N do
| s:=[op(s),y[j]];
end
ss:={};
for j to N do
| ss:=ss union {y[j]=YY[j]};
end
for k to N do
| if k = 2 then
| | print("products");
| end
| t:=combinat[choose](s,k);
| for pp in t do
| | p:=1;
| | for a in pp do
| | | p := p * a;
| | end
| | q:=subs(ss,p);
| | q:=convert(expand(q),mod2);
| | print(p,"===",q);
| end
end
```

Algorithm 8: A part of the Algorithm 5.2

```
print("INVERSE MAPPING");
s:=[];
for j to N do
 | s:=[op(s),x[j]];
end
ss:={};
for j to N do
 | ss:=ss union {x[j]=XX[j]};
end
for k to N do
 | if k=2 then
 |  | print("products");
 | end
 | t:=combinat[choose](s,k);
 | for pp in t do
 |  | p:=1;
 |  | for a in pp do
 |  |  | p := p * a;
 |  | end
 |  | q:=subs(ss,p);
 |  | q:=convert(expand(q),mod2);
 |  | print(p,"===",q);
 | end
end
print("SOLUTIONS THAT ARE LINEAR IN Y1 and X1");
col([0]); print("SOLUTION BY LINEAR SYSTEM");
syst();
End Proc:
```

Algorithm 9: A part of the Algorithm 10

```
prx:= proc(k);
Local variables i,l,p;
Global variables N; l:=binaryL(k,N);
p:=1;
for i to N do
 | if l[i] = 1 then
 |  | p := p * x[i];
 | end
end
p;
End proc
```

Algorithm 10: A part of the Algorithm 5.2

```
syst:=proc();
Local variables Z, i, j, p, px, py, pyx, s, c, k;
Global variables XX,YY,M;
s := (k[y[1]] = 1);
print("CONDITIONS");
print("MUST BE LINEAR IN Y1==¿",s);
s := s;
for i to M-1 do
    px:=prx(i);
    if has(px,x[1]) then
        c:=0;
        for j to M-1 do
            py:=pry(j);
            pyx:=pryx(j);
            if sous(pyx + Z, px)and((py = y[1])ornothas(py, y[1])) then
                c := c + k[py];
            end
        end
        if px = x[1] then
            c := (c = 1);
            else
                c := (c = 0)
            end
            ;
        end
        if c ≠ (0 = 0) then
            s := sunionc;
        end
        if px = x[1] then
            cond:="MUST CONTAIN" ;
            else
                cond:="MUST HAVE NO"
            end
            ;
        end
        print(cond, px,"===¿",c);
    end
end
```

Algorithm 11: A part of the Algorithm 5.2

```
print("THE LINEAR SYSTEM IS");
print(s);
s:=msolve(s,2);
print("THE GENERAL SOLUTION FORM IS");
print(s);
s := subs(Z1 = 0, s);
s := subs(Z2 = 0, s);
s := subs(Z3 = 0, s);
s := subs(Z4 = 0, s);
s := subs(Z5 = 0, s);
s := subs(Z6 = 0, s);
s := subs(Z7 = 0, s);
s := subs(Z8 = 0, s);
print("A PARTICULAR SOLUTION IS");
print(s);
p:=0;
for j to M-1 do
    py:=pry(j);
    c:=subs(s,k[py]);
    if c ≠ k[py] then
        p := p + c * py;
    end
end
print("SOLUTION FOR THE POLYNOMIAL LINEAR IN X1 and Y1=",p);
End Proc:
```

Algorithm 12: A part of the Algorithm 13

```
idem:=proc(p,q) ;
Local variables i,ok;
ok:=true;
for i from 2 to N do
    if (p[i] ≠ q[i]) then
        ok:=false;
    end
end
ok;
End Proc:
```

Algorithm 13: A part of the Algorithm 5.2

```
col:=proc(c) ;
Local variables i, j, k, ok;
k:=nops(c);
ok:=true;
for  i to k do
    for  j from i+1 to k do
        if  idem(X[i],X[j]) and (c[i]=c[j]) then
            ok:=false;
        end
        if  idem(Y[i],Y[j]) and (c[i]=c[j]) then
            ok:=false;
        end
    end
end
if  not ok then
    else if  k=M then
        print(fctx(c),"====",fcty(c));
        else
            col([op(c),0]);
            col([op(c),1]);
        end
    end
end
End Proc:
```

Algorithm 14: A part of the Algorithm 5.2

```
pry:=proc(k);
Local variables i, l, p;
Global variable N;
l:=binaryL(k,N); p:=1;
for i to N do
    if l[i]=1 then
        p := p * y[i];
    end
end
p:=convert(expand(p),mod2);
p;
End proc
```

Algorithm 15: A part of the Algorithm 5.2

```
pryx:=proc(k);
Local variables i, l, p;
Global variables N,YY; l:=binaryL (k,N); p:=1;  for i to N do
|  if  l[i]=1 then
|  |  p := p * YY[i];
|  end
end
p:=convert(expand(p),mod2); p;
End Proc:
```

Algorithm 16: A part of the Algorithm 5.2

```
all:=proc(f) ;
Local variables i;
Global variables M;
if  nops(f)=M then
|  test(f) else
|  |  for  i from 0 to M-1 do
|  |  |  if  not member(i,f) then
|  |  |  |  all([op(f),i]);
|  |  |  end
|  |  end
|  end
end
End Proc:
```

Algorithm 17: A part of the Algorithm 5.2

```
ex:=proc(k) ;
Local variables f,a;
Global variables M;
if  k = 0 then
|   all([]);
|   if  k=1 then
|   |   f:=[];
|   |   while nops(f) < M do
|   |   |   a:=rand() mod M;
|   |   |   if  not member(a,f) then
|   |   |   |   f:=[op(f),a];
|   |   |   end
|   |   end
|   |   test(f);
|   end
end
End Proc:
```

Algorithm 18: A part of the Algorithm 5.2

5.3 Algorithm {In situ computation over $\mathbb{Z}/N\mathbb{Z}$}

- Algorithm that compute a mapping E over $\mathbb{Z}/N\mathbb{Z}$ by a sequence of linear assignments. This algorithm takes a square matrix as input and return the corresponding assignment matrices as out put. We can construct sequence of linear assignments directly from these assignment matrices.

```
gcd–multipliers := proc(x::list, alpha::integer, i::integer);
Local variables tmp , fac, projections,l,j,k,mres;
fac := ifactors(alpha)[2];
projections := []:;
for  k from 1 to nops(fac) do
|   l := [seq(0, j=1..nops(x))];
|   l[i] := 1;
|   if igcd(x[i], fac[k][1]) ≠ 1 then
|   |   for  j from 1 to nops(x) do
|   |   |   if  igcd(x[j], fac[k][1]) = 1 then
|   |   |   |   l[j] := 1;
|   |   |   |   break;
|   |   |   end
|   |   end
|   end
|   projections := [op(projections), l];
end
tmp := [seq(fac[k,1]^fac[k,2], k=1..nops(fac))];
mres := [];
mres :=[op(mres), chrem(projections, tmp)][];
End Proc:
```

Algorithm 19: A part of the Algorithm 5.3

```
creatematrix := proc(d::integer);
Local variables M, i;
M := Matrix(d,d,0);
for  i from 1 to d do
|   M[i,i] := 1:
end
return M;
End Proc:
```

Algorithm 22: A part of the Algorithm 5.3

```
isone := proc(M::Matrix);
Local variables i, j;
for  i from 1 to RowDimension(M) do
|   for  j from 1 to ColumnDimension(M) do
|   |   if  M[i, j]! = 1 then
|   |   |   return false:
|   |   end
|   end
end
return true;
End Proc:
```

Algorithm 20: A part of the Algorithm 5.3

```
with(ListTools);
matrixop := proc(Ma::Matrix, N::integer);
Local variables i, j, k, nr, nc, left, right, g, l, T, G, M, U, final, F;
M := copy(Ma);
left := [];
right := [];
nr := RowDimension(M);
nc := ColumnDimension(M);
for k from 1 to min(nr, nc) do
    g:= igcd ( seq(M[i,k], i=1..nr) );
    if g = 1 then
        continue;
    end
    if igcd(M[k,k]/g, N) ≠ 1 then
        l := gcd–multipliers([seq(M[i,k]/g, i=1..nr)], N, k);
        T := creatematrix(nr)mod N;
        for j from 1 to nr do
            T[k,j] := l[j];
        end
        M := Multiply(T , M)mod N: left := [op(left), MatrixInverse(T)modN];
    end
    G:= creatematrix(nc);
    G[k,k] := M[k,k]: M := Multiply(M , MatrixInverse(G))mod N;
    U := creatematrix(nc);
    for j from 1 to nc do
        U[k,j] := M[k,j];
    end
    M := Multiply(M , MatrixInverse(U))mod N;
    right := [op(right), Multiply(U , G)mod N ];
end
final := [];
for i from 1 to nops(left) do
    if isone(left[i]) = false then
        final := [op(final), left[i] ];
    end
end
for i from nops(right) to 1 by -1 do
    if isone(right[i]) = false then
        final :=[op(final), right[i]];
    end
end
Reverse(final);
End Proc:
```

Algorithm 21: A part of the Algorithm 5.3

5.4 Algorithm:

- Algorithm that compute a mapping $E : \mathbb{Z}^m \longrightarrow \mathbb{Z}^m$, $m > 2$, where $\mathbb{Z}$ is a set of integers, defined as

$$E : (x_1, x_2, x_3, \cdots, x_n) = \begin{pmatrix} F_1x_1 + F_2x_2 + F_3x_3 + \ldots + F_mx_n \\ F_2x_1 + F_3x_2 + F_4x_3 + \ldots + F_{m+1}x_n \\ F_3x_1 + F_4x_2 + F_5x_3 + \ldots + F_{m+2}x_n \\ \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \\ F_nx_1 + F_{n+1}x_2 + F_{n+2}x_3 + \ldots + F_{2n-1}x_n \end{pmatrix}$$

such that $F_n := F_{n-1} + F_{n-2}$, $\forall n \in \mathbb{Z}$, can be computed with $m + 2$ number of linear assignments. The algorithm takes only 2 initial entries of the square matrix A at positions a_{11} and a_{12} and construct all other coefficients and return the linear assignments that can compute the mapping E.

Input: integer
Output: fibonacci numbers
oo:= **proc(n)**;
local variables i, f, s;
s:=[1];
$f[1] := 1; f[2] := 1;$
for *i from 2 to n* **do**
  $f[i + 1] := f[i] + f[i - 1];$
  $s := [op(s), f[i + 1]];$
end
End proc;

Algorithm 23: a part of the Algorithm

```
Input: integer
Output: assignments
ass:= proc(N);
local variables j, jj, ii, xxx, first, second, m, p, pp, s, xx;
p:=0; pp:=0; first:=2; second:=3 ;
if N ≤ 2 then
    print("N must be greater than 2");
    else
        s:=oo(N);
        for j from 2 to N do
            p:=p+s[j-1]*x[j];
        end
        xx[1]:=x[1]+p;
        for jj from 2 to N-1 do
            pp:=pp+s[jj-1]*x[jj+1];
        end
        print(x[1],"=",xx[1]);
        xx[2]:=xx[1]+x[2]+pp;
        print(x[2],"=",x[1]+x[2]+pp);
        xx[3]:=first*xx[1]+second*xx[2];
        print(x[3],"=",first*x[1]+second*x[2]);
        if N > 3 then
            xx[4]:=xx[3]+((second-first)*xx[1]+first*xx[2]);
            print(x[4],"=",x[3]+(second-first)*x[1]+first*x[2]);
            if N > 4 then
                for iii from 5 to N do
                    xx[iii]:=xx[iii-2]+xx[iii-1];
                end
            end
        end
        for v from 5 to N do
            print(x[v],"=",x[v-2]+x[v-1]);
        end
        print(x[1],":=",x[3]-((second-first)*(x[1])+(first)*(x[2])));
        print(x[2],":=",x[3]-x[1]); print("******************************");
        xxx[1]:=xx[3]-((second-first)*(xx[1])+(first)*(xx[2])); xxx[2]:=xx[3]-xxx[1];
        print(x[1]=xxx[1]); print(x[2]=xxx[2]);
        for ii from 3 to N do
            print(x[ii]=xx[ii]);
        end
    end
end
End;
```

5.5 Conclusions:

The *in situ* computation of mapping by a sequence of assignments, without using any extra variable other than the variables available as input variables, is a new way in code optimization according to the best of our knowledge. This way of computation helps directly to reduce the memory usage and ultimately the idea can be implemented in compiler/processor optimization, register allocation and chip designing.
The sequential decomposition of matrices is a powerful tool for computations of linear transformations using minimal memory. We prove that, for such computation extra memory than input variables is not required. We prove the sequential decomposition of linear mapping over field and extended this idea over the Ring and set of integers. It is still an open path for the researchers to improve the efficiency of these algorithms and to find the minimum number of assignments required to compute a linear mapping using *in situ* computation. The parallel applications of this work, in the decomposition of matrices may lead to solve system of linear equations, to triangularize a matrix efficiently etc. Since the matrices can be triangulate with different methods, therefore we can find the applications of *in situ* computations of mappings corresponding to these methods. For example, Triangularization of a square matrix, using Extended Euclidean Algorithm leads to find the sequence of linear assignments that can compute linear mappings using *in situ* program, consists of these linear assignments. The upper bound, of Extended Euclidean Algorithm, that has been found by Lame theorem can be helpful to find the number of assignments used by corresponding *in situ* program, and the lower bound will yield a new interesting results both for such algorithms and *in situ* programs.

We have also described that *in situ* computation of general and bijective boolean mappings and presented a suitable bound for this computation. We introduce a new method on the basis of bijective boolean mappings via algebraic operations over polynomials in $GF2$. This new method is a powerful tool to compute *in situ* boolean mappings. Initially, this method compute linear boolean polynomials that are linear both in x_1 and y_1.
The main motivation of the work presented is the processor performances because the application of this work concerns to hardware as well as software in the sense of chip designing and compiler optimization.

Bibliography

[1] W. Abu-Sufah. Improving the Performance of Virtual Memory Computers. PhD thesis, University of Illinois at Urbana-Champaign, Nov 1978.

[2] O. Azizi, J. Collins, D. Patil, Hong Wang, and M. Horowitz. Processor Performance Modeling using Symbolic Simulation. *ISPASS 2008, IEEE International Symposium on Performance Analysis of Systems and software*, pages 127–138, April 2008.

[3] Richard Beigel. The polynomial method in circuit complexity. In *Structure in Complexity Theory Conference*, pages 82–95, 1993.

[4] Florent Bouchez, Alain Darte, Christophe Guillon, and Fabrice Rastello. Register Allocation: What does the NP-completeness Proof of Chaitin et al. Really Prove? In *Fifth Annual Workshop on Duplicating, Deconstructing and Debunking (WDDD2006) (at ISCA-33)*, June 2006. `http://perso.ens-lyon.fr/fabrice.rastello/Biblio_Perso/Articles/WDDD06.pdf`.

[5] Preston Briggs, Keith D. Cooper, and Linda Torczon. Improvements to graph coloring register allocation. *ACM Trans. Program. Lang. Syst.*, 16(3):428–455, 1994.

[6] A. E. Brown, J. P. Eckhardt, M. D. Mayo, W. A. Svarczkopf, and S. P. Gaur. Improved performance of IBM Enterprise System/9000 bipolar logic chips. *IBM J. Res. Dev.*, 36(5):829–834, 1992.

[7] Serge Burckel. Closed Iterative Calculus. In *Theoretical Computer Science*, pages 371–378, 1996.

[8] Serge Burckel and E. Gioan. In situ design of register operations. In *Proceedings of ISVLSI08*, pages 287–292, 2008.

[9] Serge Burckel and M. Morillon. Three Generators for Minimal Writing-Space Computations. In *Theoretical Informatics and Application*, volume 34, pages 131–138, 2000.

[10] Serge Burckel and M. Morillon. Quadratic Sequential Computations of Boolean Mappings. In *Theory of Computing Systems. 37(4)*, pages 519–525, 2004.

[11] Serge Burckel and M. Morillon. Sequential computation of linear boolean mappings. In *Theoretical Computer Science, (314)*, pages 287–292. Springer, 2004.

[12] Hand Out Written by Maggie Johnson. Code optimization, November 26, 2007. `http://www.ecs.syr.edu/faculty/mccracken/cis631/materials/`.

[13] IBM Systems Information Center. Compiler optimization techniques. `http://publib.boulder.ibm.com/infocenter/systems`.

[14] G. J. Chaitin. Register allocation & spilling via graph coloring. In *SIGPLAN '82: Proceedings of the 1982 SIGPLAN symposium on Compiler construction*, pages 98–105, New York, NY, USA, 1982. ACM.

[15] Gregory J. Chaitin, Marc A. Auslander, Ashok K. Chandra, John Cocke, Martin E. Hopkins, and Peter W. Markstein. Register allocation via coloring. *Comput. Lang.*, 6(1):47–57, 1981.

[16] Siddhartha Chatterjee, Vibhor V. Jain, Alvin R. Lebeck, Shyam Mundhra, and Mithuna Thottethodi. Nonlinear array layouts for hierarchical memory systems. In *In Proceedings of the 1999 ACM International Conference on Supercomputing*, pages 444–453, 1999.

[17] A. Church. An unsolvable problem in elementary number theory. *American Journal of Mathematics*, 58:356–363, 1936.

[18] James Cohoon and Jack Davidson. *C++ Program Design: An Introduction to Programming and Object-Oriented Design*. Irwin McGraw-Hill, Boston, Massachusetts, 1997.

[19] Y. Crama and Peter L. Hammer. *Boolean Functions - Theory, Algorithms, and Applications*. In preparation, December 02, 2008.

[20] J. Daintith. A Dictionary of Computing, 2004. `http://www.oxfordreference.com`.

[21] Patel S. Friendly, D. and Y Patt. Putting the fill unit to work: dynamic optimizations for trace cache microprocesors. In *Proceedings of the 31st International Symposium on Microarchitecture (MICRO-31)*, pages 173–181, 1998.

[22] Matteo Frigo, Charles E. Leiserson, Harald Prokop, Sridhar Ramachandran, and Z W(l. Cache-oblivious algorithms (extended abstract). In *In Proc. 40th Annual Symposium on Foundations of Computer Science*, pages 285–397. IEEE Computer Society Press, 1999.

[23] Lal George and Andrew W. Appel. Iterated register coalescing. *ACM Trans. Program. Lang. Syst.*, 18(3):300–324, 1996.

[24] G. H. Golub and C. F. Van Loan. Matrix Computations. *Johns Hopkins University Press*, 1989.

[25] F. Havet. *Graph colouring and applications*. Habilitation à diriger des recherches, Université de Nice-Sophia Antipolis, December 12 2007.

[26] Vincent Heuring and Harry Jordan. *Computer Systems Design and Architecture*.

[27] K. Hoste and L. Eeckhout. Cole: Compiler optimization level exploration. In *the International Symposium on Code Generation and Optimization (CGO)*, 2008.

[28] R. Ismail and V.M. Rooney. *Microprocessor Hardware and Software Concepts*. Macmillan, Collier Macmillan, New York, London, 1987.

[29] S. Hammarling J. Dongarra, J. Du Croz and I. Duff. A set of level 3 basic linear algebra subprograms. *ACM Transactions on Mathematical Software*, pages 1–17, March 1990.

[30] John Jannotti. Ars Technicia: The PC Enthusiasts Resource, 2001.

[31] U. Meier K. Gallivan, W. Jrdby and A. Sameh. The impact of hierarchical memory systems on linear algebra algorithm design. Technical Report, University of Illinios, 1987.

[32] J. Keller. The 21264: a superscalar alpha processor with out-of-order execution. *9th Annual Microprocessor Forum*, 1996.

[33] S. C. Kleene. λ-definability and recursiveness. *Duke Mathematical Journal*, 2:340–353, 1936.

[34] D. König. Über Graphen und ihre Anwendung auf Determinantentheorie und Mengenlehre. *Mathematische Annalen*, 77:453–465, 1916.

[35] A. Kumar. The HP PA-8000 RISC CPU: A high performance out-of-order processor. In *Hot Chips VIII*, 1996.

[36] Monica S. Lam, Edward E. Rothberg, and Michael E. Wolf. The cache performance and optimizations of blocked algorithms. In *In Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 63–74, 1991.

[37] V. K. Leont'ev. Certain problems associated with boolean polynomials. *Computing Center, Russian academy of Sciences, ul. Vavilova 40, Moscow, GSP-1, 117967 Russia*, September 11, 1998.

[38] E. E. Rothberg M. S. Lam and M. E. Wolf. The cache performance and optimization of blocked algorithms. In *The Sixth International Conference on Architectural Support for Programming Languages and Operating Systems*, April 1991.

[39] F. J. MacWilliams and N. J. A. Sloane. *The theory of error correcting codes / F.J. MacWilliams, N.J.A. Sloane*. North-Holland Pub. Co. ; sole distributors for the U.S.A. and Canada, Elsevier/North-Holland, Amsterdam ; New York : New York :, 1977.

[40] A. C. McKeller and E. G. Coffman. The organization of matrices and matrix operations in a paged multi-programming environment. *CACM*, 12(3):153–165, 1969.

[41] Joon-Sang Park Michael, Michael Penner, and Viktor K Prasanna. Optimizing graph algorithms for improved cache performance. In *In Proc. Intl Parallel and Distributed Processing Symp. (IPDPS 2002), Fort Lauderdale, FL*, pages 769–782, 2002.

[42] R.C. Miller and B. J. Oldfield. Producing Computer Instructions for the PACT I Compiler. *Journal of the ACM*, 1956.

[43] John Miranda. The Usage of Compiler Optimization by Programmers. In *Thesis, School of Engineering and Applied Science University of Virginia*, 2001.

[44] Neungsoo Park, Dongsoo Kang, Kiran Bondalapati, and Viktor K. Prasanna. Dynamic data layouts for cache-conscious factorization of dft. In *In Proc. of International Parallel and Distributed Processing Symposium*, pages 693–701, 2000.

[45] Bennett S. Rotenberg, E. and J. E. Smith. Trace cache: a low latency approach to high band- width instruction fetching. In *29th International Symposium on Microarchitecture*, pages 24–34, 1996.

[46] S. Rudeanu. On the decomposition of boolean functions via boolean equations. *j-jucs*, 10(9):1294–1301, 2004. `http://www.jucs.org/jucs_10_9/on_the_decomposition_of`.

[47] S. Rus and L. Rauchwerger. *Compiler technology for migrating sequential code to multithreaded Architectures*. Technical Report, Texas A and M University, College Station, Texas, USA, 2006.

[48] K. S. McKinley S. Carr and C. Tseng. Compiler optimization for improving data locality. *Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems*.

[49] R. Muth S. Debray, W. Evans and B. D. Sutter. Compiler techniques for code compaction. *ACM Trans. on Programming Languages and Systems, 22(2)*, pages 378–415, 2000.

[50] Vivek Sarkar. Code optimization of parallel programs: evolutionary vs. revolutionary approaches. In *CGO '08: Proceedings of the sixth annual IEEE/ACM international symposium on Code generation and optimization*, pages 1–1, New York, NY, USA, 2008. ACM.

[51] Vivek Sarkar. Challenges in code optimization of parallel programs. In *CC*, page 1, 2009.

[52] Ian L. Sayers, Adrian P. Robson, Alan E. Adams, and E. Graeme Chester. *Principles of microprocessors*. CRC Press, Inc., Boca Raton, FL, USA, 1991.

[53] P. B. Schneck. A Survey of Compiler Optimization Techniques. In *Proceedings of ACM Conference*, pages 106–113, 1973.

[54] D. S. Scott and C. Strachey. Towards a mathematical semantics for computer languages. 1971.

[55] Michael Slater, editor. *A guide to RISC microprocessors*. Academic Press Professional, Inc., San Diego, CA, USA, 1992.

[56] R. M Tomasulo. An efficient algorithm for exploiting multiple arithmetic units. *IBM Journal of Research and Development*, pages 25–233, 1967.

[57] A. Turing. Computability and λ-definability. *Journal of Symbolic Logic*, 2:153–163, 1937.

[58] R. Clint Whaley and Jack J. Dongarra. Automatically tuned linear algebra software. In *Supercomputing '98: Proceedings of the 1998 ACM/IEEE conference on Supercomputing (CDROM)*, pages 1–27, Washington, DC, USA, 1998. IEEE Computer Society.

[59] M. E. Wolf and S. Lam. M. A data locality optimizing algorithm. In *The SIGPLAN'91 Conference on Programing Language Design and Implementation*, pages 30–44, June 1991.

[60] M. J. Wolfe. More iteration space tiling. In *Supercomputing'89*, November 1989.

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399